Enger, Sebastian

Softwareseitige Absicherung von Linux-Systemen gegen netzwerkbasierende Angriffe

Bachelorarbeit

HOCHSCHULE MITTWEIDA (FH)

UNIVERSITY OF APPLIED SCIENCES

Fachbereich Informationstechnik & Elektrotechnik

Mittweida, 2005

Enger, Sebastian

Softwareseitige Absicherung von Linux-Systemen gegen netzwerkbasierende Angriffe

eingereicht als

Bachelorarbeit

an der

HOCHSCHULE MITTWEIDA (FH)

--

UNIVERSITY OF APPLIED SCIENCES

Fachbereich Informationstechnik & Elektrotechnik

Mittweida, 2005

Erstprüfer: Prof. Dr.-Ing. habil. Joachim Geiler

Zweitprüfer: Dipl.-Inf. Matthias Lühr (FH)

vorgelegte Arbeit wurde verteidigt am: 07.10.2005

Enger, Sebastian:
Softwareseitige Absicherung von Linux-Systemen gegen netzwerkbasierende Angriffe. - 2005. -
108 S.
Mittweida, Hochschule Mittweida (FH), Fachbereich Informationstechnik & Elektronik,
Bachelorarbeit, 2005

Referat:
Ziel der vorliegenden Bachelorarbeit ist es, eine Anleitung zu entwickeln, mit der es möglich ist, ein Linuxsystem gegenüber vorgestellten Angriffsarten softwareseitig zu schützen. Die dabei verwendeten Werkzeuge werden vorgestellt und anschließend aufgezeigt, welche Angriffe gegen das Linuxsystem gerichtet werden können und wie man sich dagegen schützen kann. Weiterhin werden allgemeine Abwehrmaßnahmen in Netzwerken erläutert und auf spezielle Sicherheitsmechanismen und Sicherheitspatches eingegangen. Im anschließenden Abschnitt wird aufgezeigt, wie man mit kostengünstigen Mitteln einen Linuxrechner gegen Angriffe härten kann. Konzentriert werden soll sich im Rahmen der Bachelorarbeit ausschließlich auf die hier vorgestellten Angriffe und Verteidigungsmechanismen, da eine noch ausführlichere Ausarbeitung den Rahmen der Bachelorarbeit bei weitem überschritten hätte.

Ehrenwörtliche Erklärung:

Ich erkläre, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Bearbeitungsort, Datum Unterschrift

Abbildungsverzeichnis

Inhaltsverzeichnis

1 Einleitung

Eine Umfrage von Ernst und Young ergab, dass Unternehmen weltweit an der IT-Sicherheit sparen und sich damit selbst in Bedrängnis bringen. Ein Drittel der 1400 befragten Verantwortlichen gaben an, dass sie im Falle eines Angriffes nur unzureichend in der Lage seien, auf den Vorfall zu reagieren. Erschreckend hierbei ist die Tatsache, dass immerhin 34 % der befragten Unternehmen nicht eindeutig beurteilen können, ob ihr Unternehmen gerade angegriffen wird. Dazu kommt der Fakt, das 60 % der Unternehmen den Wert des Return on Investment für IT-Sicherheitsaspekte kaum oder gar nicht mit einkalkulieren. [1]
Anhand dieser Umfrage von Ernst und Young kann man sehr leicht erkennen, dass ein Großteil der IT-Unternehmen das Thema IT-Sicherheit nur unzureichend beurteilen und somit sich selbst einer größeren Gefahr als nötig aussetzen.
Im weiteren Verlauf der Arbeit wird kurz auf den Sinn und Zweck der IT-Sicherheit, wichtiger Täterprofile, Angriffsarten, Informationssysteme zur IT-Sicherheit und Sicherheitswerkzeuge eingegangen. Danach werden bestimmte, ausgewählte Angriffsmethoden beschrieben und Abwehrmöglichkeiten für diese speziellen Angriffe aufgezeigt. Im Anschluss daran geht man allgemein auf Abwehrmaßnahmen im Netzwerk ein, die mit den bereits beschriebenen Maßnahmen kombiniert werden können, um die Gesamtsicherheit des Linuxsystems weiter zu erhöhen. Danach schließt sich das Kapitel "Sicherheitsmechanismen für Linuxsysteme" an, in welchem detailliert auf aktuelle Mechanismen zur Erhöhung der Systemsicherheit eingegangen wird und diese mit ihren Vor- und Nachteilen tabellarisch gegenübergestellt werden. Weiterhin wird aufgezeigt, welche Software für welche Art von Sicherheitsbedürfnis geeignet ist. Dabei werden die Produkte SE-Linux, Bastille-Linux, LIDS, Grsecurity und Openwall vorgestellt. Anschließend wird anhand dieser Mechanismen ein sicherer Linux Server implementiert und aufgezeigt, welche Vorteile dieses System gegenüber einem ungeschützten System besitzt.

1.1 Sicherheit in IT-Systemen von Unternehmen

Unternehmen haben verschiedene Anforderungen an die IT-Systeme. Im Großen und Ganzen lassen sich dabei folgende Aspekte zusammenfassen, die immer wieder im Mittelpunkt dieser Betrachtung liegen. Solche Anforderungen beziehen sich auf die entsprechende Verfügbarkeit von IT-Systemen sowie die entsprechende Qualität der zu nutzenden Software. Für Unternehmen ist in diesem Zusammenhang auch der Fakt der Wirtschaftlichkeit solcher Systeme wichtig. Weiterhin spielt na-

türlich der Aspekt der IT-Sicherheit sowie der Vertraulichkeit, also der Schutz vor Kenntnisnahme Dritter, der Integrität der Daten sowie der Authentiziät, also der Schutz vor dem Verändern der Daten, in Unternehmen eine Rolle. [2]
Diese Anforderungen stehen teilweise im konträren Gegensatz zur Investitionspolitik der Unternehmen. Trotz verstärkter Viren und Würmerangriffe in den letzten Monaten sind viele Unternehmen nicht gewillt, in die IT-Sicherheit ihrer eigenen Unternehmen zu investieren. Laut [3] wollen 68 % der befragten Firmen ihre "Ausgaben einfrieren oder gar senken". [3] Neben den Angriffen durch Viren und Würmer spielen auch die so genannten Denial of Service Angriffe eine immer größer werdende Rolle in Unternehmen. Diese haben zum Ziel, den entsprechenden Rechner lahm zu legen, indem tausendfache, sinnlose, parallele Anfragen an die Rechner gestellt werden.[3] Weiterhin scheinen solche Angriffe in letzter Zeit verstärkt gegen Wettbüros wie z.B. mybet.com eingesetzt zu werden, mit dem Ziel der Erpressung von Geld. [4]
Heutzutage lässt sich die Sicherheit im eigenen Unternehmen auch leicht selbst austesten, indem man verschiedene, so genannte Security Scanner gegen die eigenen Internetrechner einsetzt und diese damit auf Sicherheitslücken testet. Aus dem Bereich Open Source ist auf jeden Fall das Programm Nessus zu erwähnen, auf das später noch genauer eingegangen wird. Ein sehr bekanntes kommerzielles Produkt ist der Internet Security Scanner von Internet Security Systems. Dieser kostet jedoch mehrere Tausend Euro. Mit Hilfe solcher Programme kann ein Unternehmen leicht selbst testen, gegen welche Art von Angriffen das eigene System nicht ausreichend geschützt ist und man kann entsprechende Gegenmaßnahmen treffen. Leider wird kostenlosen Programmen oft nicht genug vertraut und auf den Einsatz solcher Software verzichtet. Andererseits ist der kommerzielle Internet Security Scanner sehr teuer und so wird diese Investion oft gescheut, da der Return on Investment meist nicht erkannt wird. So stehen Unternehmen dann auch oft ratlos da, wenn sie angegriffen werden, wie die Studie von Ernst und Young gezeigt hat. [1]

1.2 Sinn und Zweck der IT-Sicherheit

Sinn der Ausarbeitung einer unternehmensweiten Sicherheitspolitik ist es, das Unternehmen vor unbefugten Zugriff durch Dritte zu schützen. Dazu wird nach einem Prozessmodell eine Richtlinie entwickelt, die auf die Aspekte Mensch, Strategie und Technik basiert. Mittels dieser Richtlinie werden dann Mitarbeiter geschult, Sicherheitshardware wie Firewalls gekauft und entsprechende Strategien zur Risikominimierung, Risikoverhinderung und Leitfäden für das Vorgehen im Einbruchsfall entwickelt und umgesetzt. Primär dient die IT-Sicherheit dem Ziel der Verhinderung von

Angriffen auf das eigene Unternehmen, sekundär mit den Protokollen, die durchgeführt werden, wenn es zu einem Einbruch kam und wie dieser aufgearbeitet wird. Auf diese Aspekte soll jedoch im Rahmen dieser Arbeit nicht weiter eingegangen werden.

1.3 Täterprofile

Üblicherweise wird unter dem Begriff Hacker ein Mensch verstanden, der in ein fremdes Computersystem eindringt und dort Schaden anrichtet. Man muss hierbei zwischen "Hackern", "Script Kiddies", und "Crackern" unterscheiden. Hacker sind meist sehr klevere Programmierer, die sich aus eigenem Interesse mit der Thematik Sicherheitslücken in Computersystemen auseinandersetzen. Script Kiddies sind meist junge Täter, die vorhandene Programme benutzen, um damit Schaden anzurichten. Cracker wiederum sind Personen, die aus finanziellem Interesse in Systeme eindringen oder aber anderweitige Vorteile für sich daraus ziehen wollen. Personen, die Zugriff oder Wissen über die innere Struktur eines Unternehmens haben, nennt man Insider. Oft sind dies ehemalige Mitarbeiter, die der Firma Schaden zufügen wollen, weil sie z.B. gekündigt worden sind. Neben den angesprochenen Täterprofilen gibt es auch die professionelle Wirtschaftsspionage, deren Ziel es ist, an die Geheimnisse einer Firma zu kommen, um sich damit einen Vorteil auf dem Markt zu verschaffen. [5]

1.4 Angriffsarten

Es gibt verschiedene Angriffsarten auf moderne Computersysteme. Die wichtigsten Angriffsarten sind Angriffe über das Netzwerk, Social Engineering und die Umgehung von physischen Sicherheitsmaßnahmen. In den Bereich der Informationsbeschaffung fällt auch das so genannte Trashing, also das Durchsuchen des firmeneigenen Mülls nach verwertbaren Dokumenten.
Angriffe über das Netzwerk werden mit Hilfe von Programmen ausgeführt, die verschiedene Netzwerkprotokolle und Netzwerkapplikationen nutzen. Meistens werden dazu Schwachstellen in Implementationen genutzt, um Angriffe über das Netzwerk auf Internetrechner zu starten. Beispiele für diese Angriffe sind das IP-Spoofing, Hijacking von Verbindungen, Denial of Service Angriffe oder das Mitschneiden von Kommunikationsbeziehungen. Unter dem Begriff "Social Engineering" versteht man das Ausnutzen des Faktors Mensch bei der IT-Sicherheit. Angreifer geben sich z.B. als Administrator aus und verlangen von Mitarbeitern, das eigene Passwort herauszugeben, da man an ihrem Rechner neue Software installieren müsse. Diese Form des Angriffes läßt sich durch ent-

sprechende Schulung von Mitarbeitern vermeiden. Physische Sicherheitsmaßnahmen bilden den Kern der eigenen Sicherheitspolitik. Ein Internet Service Provider wie die Telekom muss die eigenen Rechner und Router durch Zugriffskontrollen, Videoüberwachungen und redundante Hardware vor Ausfall schützen. Wenn diese Schutzmaßnahmen umgangen werden können, so kann es leicht zu einem Angriff auf das Unternehmen kommen.[5]

1.5 Informationssysteme zur IT-Sicherheit

Bundesamt für Sicherheit

Das Bundesamt für Sicherheit ist unter der Adresse [6] zu erreichen und hat es sich zur Aufgabe gemacht, "Sicherheitsrisiken bei der Anwendung der Informationstechnik" aufzufinden und entsprechende Vorkehrungen zu entwickeln, die diese Sicherheitsrisiken einschränken. Weiterhin informiert das BSI über aktuelle Sicherheitsvorfälle, wie aktuelle Viren und Würmer, und prüft auch bestehende IT-Systeme auf Sicherheit. Weiterhin hat das BSI eine Beratungsfunktion inne, die es dem Amt erlauben, Hersteller und Produzenten von Sicherheitshard- und Software entsprechend zu beraten. Weiterhin bietet das BSI eine ganze Reihe von Publikationen zum Thema IT-Sicherheit, die kostenlos eingesehen werden können. [6]

Securityfocus.com

Die Internetseite Securityfocus.com ist laut Aussage von Kelly Martin eine der umfangreichsten und verlässlichsten Quellen zum Thema Sicherheit im Internet. Dabei versucht die Internetseite herstellerunabhängig zu agieren und objektive und aktuelle Informationen für alle sicherheitsinteressierten Personen zu bieten. Im Bereich Security Focus News berichten Sicherheitsexperten über aktuelle Vorfälle und Vorkommnisse im Bereich IT-Sicherheit und im Abschnitt Securityfocus.com Vulnerability Database werden aktuelle Schwachpunkte in Soft- und Hardware beleuchtet, oft mit einem Beispielangriff, Exploit genannt, untermauert, an dem man den schwachen Punkt der Software nachvollziehen kann. In diesem Zusammenhang bleibt zu erwähnen, dass BugTraq eine Mailingliste zum Thema Sicherheit ist, die sehr stark frequentiert wird. [7]

Cert Coordination Center

Das Cert Coordination Center ist eine Anlaufstelle für Probleme bei dem Thema Internetsicherheit. Fachleute analysieren aktuelle Schwachstellen in Soft- und Hardwaresystemen und bieten meist entsprechende Informationen an, wie sich diese Sichereitslücken schließen lassen. Weiterhin veröffentlicht das CERT technische Informationen zum Thema und bietet auch Trainingskurse an. Es wird von der Carnegie Mellon Universität in den USA unterhalten. [8]

2 Sicherheitswerkzeuge

Sicherheitsanalysen bilden einen Schwerpunkt beim Gesamtprozess IT-Sicherheit. Hier wird mit Hilfe von verschiedenen Werkzeugen die eigene Sicherheit geprüft, um anschließend Verbesserungen an der eigenen Konfiguration vorzunehmen. In diesem Zusammenhang steht auch die schon bereits angesprochene Selbsteinschätzung, die Teil der Sicherheitsanalyse ist. Die Sicherheitsanalyse kann man von Sicherheitsfirmen wie der netplace Telematic GmbH durchführen lassen oder selbst durchführen. Der Einsatz fremder Firmen ist jedoch entsprechend teuer, so dass viele kleinere Firmen diese Investition scheuen werden und stattdessen auf entsprechende Werkzeuge zurückgreifen und selbst die Sicherheit ihrer Software überprüfen werden. Die im Folgenden vorgestellten Programme Nemesis, Nmap und Nessus sind für dieses Vorhaben sehr gut geeignet, weshalb im Rahmen dieser Arbeit darauf eingegangen werden soll.

2.1 Paketgenerator Nemesis

Das Programm Nemesis versteht sich als Kommandozeilentool zum Erstellen und Versenden von Paketen auf Basis der verschiedenen Netzprotokolle. Es unterstützt die Protokolle IP, TCP, UDP, ICMP, ARP, DNS, ETHERNET, IGMP, IP, OSPF und RIP. Es kann verwendet werden, um die eigene Firewall, das Intrusion Detection System, den IP Stack oder weitere Dinge auszutesten. Da es ein kommandozeilenorientiertes Tool ist, kann man damit sehr gut scripten. Nemesis ist hervorragend dazu geeignet, IP Paktete mit den verschiedensten Optionsflags zu generieren. Mittels des Befehls "nemesis ip help" bekommt man alle möglichen Optionseinstellungen zum Generieren von Paketen. Da das Programm sehr umfangreich ist, wird empfohlen, das eigentliche Manuel zu lesen, um somit einen besseren Einblick in das Werkzeug zu bekommen. [9]
Die im Quellenverzeichnis gelisteten Programme sollten verwendet werden, um einen besseren

Einblick in die Art der Bedienung und die Funktionsweise von Nemesis zu bekommen. Deshalb soll hier auf eine detailliertere Beschreibung verzichtet und darauf hingewiesen werden, die im Anhang gelisteten Programme zu benutzen.

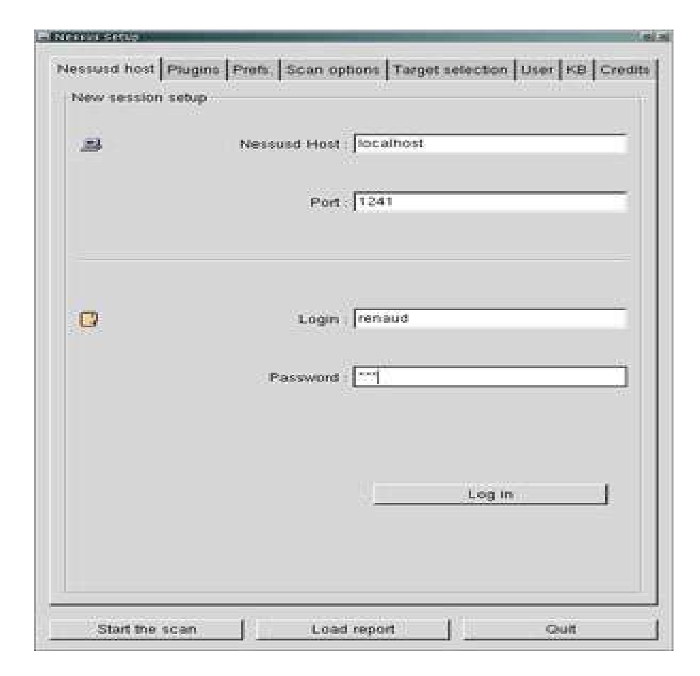
2.2 Portscanner Nmap

Nmap ist ein Portscanner, dessen Aufgabe es ist, ein Internetrechner auf offene Ports hin zu testen. Dazu nutzt der Scanner verschiedene Techniken wie den TCP-Connect()-Scan, TCP-SYN-Scan, UDP-Scan und ICMP-Scan, um nur einige zu nennen. Auf die jeweils wichtigsten Scan-techniken werde ich im Abschnitt "Netzwerkangriffe" genauer eingehen und möchte diese hier nur kurz nennen. Um den betreffenden Rechner auf offene Ports hin zu testen, sendet Nmap ent-sprechend präparierte Pakete an den Zielhost und wertet die Antwortpakete entsprechend aus und entscheidet dann, ob ein Port offen ist oder nicht. Desweiteren gibt es eine gewisse Anzahl an Möglichkeiten, die wahre Herkunft von solchen Scanversuchen zu verschleiern. Es gibt z.B. die Möglichkeit, dass man den verschickten Paketen eine bestimmte IP-Adresse zuweist und so nicht die Adresse vom scannenden Rechner angeben muss. Weiterhin gibt es die Möglichkeit des so genannten "decoy"-Scanning, bei dem man mehrere Absendeadressen einträgt, sodass es auf dem zu scannenden Host so aussieht, als ob man von mehreren Rechnern gleichzeitig gescannt wird. [10]

2.3 Sicherheitsscanner Nessus

Nessus ist ein Sicherheitsscanner, der kostenlos unter [11] zu beziehen ist. Er ist als Open Source Programm in den Software Quellen zugänglich, sodass man den Code einsehen und wenn nötig, selbst verändern kann. Der Scanvorgang läuft folgendermaßen ab: man verbindet sich mit dem Nessus-Client zu einem Nessus-Server und authentifiziert sich gegenüber diesem mittels Login und Passwort. Dann wählt man die Scaneinstellungen und den Zielhost aus und startet den Scan.
Nach Abschluss wird eine Zusammenfassung über mögliche Schwachstellen aufgezeigt und der entsprechende Report als HTML, ASCII-Text, Latex oder PDF ausgegeben.
Die Featurevielfalt zeichnet Nessus aus, denn neben aktuellen Plugins zum Scannen nach lokalen und entfernten Schwachstellen bei Internetrechnern bietet Nessus eine eigene Scripting Sprache, die es erlaubt, schnell eigene Plugins zu programmieren und sie den Nessus-Usern zur Verfügung zu stellen. Ein weiterer großer Vorteil ist es, dass Nessus eine Open Source Entwicklung ist, an der mehrere Entwickler mitwirken, sodass seine Aktualität gewahrt wird und man beim Scannen nach

Schwachstellen immer die neueste Software verwendet. [11]



aus [11]

Abbildung 2.1: Nemesis Client

3 Netzwerkbasierende Angriffe

3.1 Angriffe auf Protokollebene

3.1.1 IP-Spoofing

Die am meisten genutzte Angriffsart für weitere Angriffe auf Linuxrechner ist das Internet Adress Spoofing oder kurz IP-Spoofing genannt. Dabei werden mit Hilfe von Paketgeneratoren synthetische Datenpakete erzeugt, bei denen die Optionsflags (z.B. beim TCP-Paket SYN-Flag gesetzt) entsprechend gesetzt sind. Diese erstellten Datenpakete sind meistens mit gefälschter IP-Sendeadresse generiert worden, um so die Herkunft des Paketes zu verheimlichen. Problematisch wird es, wenn mittels IP-Spoofing Firewalls attackiert werden, die nur einen Paketfilter besitzen und somit auf Basis einfacher Filterregel bestimmen, ob das ankommende Paket zugelassen wird oder nicht.

Hat das Paket eine Absendeadresse, die im Bereich des Paketfilters liegt, wird das Paket von dem Paketfilter als regulär aus dem eigenen Adressbereich stammend angesehen und darf die Firewall passieren. Daraus resultiert, dass Sicherheitsrichtlinien nicht auf Paketfilter basieren sollten bzw.

Paketfilter nicht den hauptsächlichen Sicherheitsaspekt der eigenen Sicherheitsstrategie ausmachen. Wenn es nun gelingt, einen IP-Spoofing Angriff erfolgreich durchzuführen, kann dies als Ausgangsbasis für weitere Angriffsvarianten auf TCP, ICMP genutzt werden. Gelingt es einem Angreifer, gespoofte Pakete in das eigene Netzwerk einzuschleusen, so kann das als ein Anzeichen für einen bald folgenden Angriff gesehen werden. [12]
Mit dem vorgestellten Paketgenerator Nemesis kann man sehr leicht verschiedene Pakete generieren und deshalb soll hier auf die im Quellenverzeichnis gelisteten Programme hingewiesen werden, da man damit einen IP-Spoofing Angriff zu Testzwecken nachstellen kann.

3.1.2 IP-Fragmentierungsangriffe

Die Fragmentierung von IP Paketen wurde ursprünglich eingeführt, um Netzwerkschnittstellen zu überwinden, die nur eine festgelegte Paketlänge unterstützen, wobei das eigentlich zu übertragene Paket aber viel länger ist. Beim Ethernet beträgt die maximale Größe, die ein Paket haben kann, 1500 Bytes, jedoch kann ein IP Datagramm 65 535 Bytes lang sein. Um dieses Paket trotzdem über die Ethernetschnittstelle übertragen zu können, muss ein Router oder Gatewayrechner das Paket in Stücke zu 1500 Bytes fragmentieren und dann an den Zielhost weitersenden. Am Zielort werden mittels TCP/IP Stack die einzelnen fragmentierten Pakete wieder zu einem ganzen Paket zusammengesetzt. Jedes fragmentierte Paket enthält wichtige Statusinformationen im IP-Header. Dazu gehören Identifikationsnummer, Fragmentierungsflags, sowie ein Fragment Offset, wodurch die Identifikation und Reihenfolge der Fragmente wieder bestimmbar ist. Für Firewalls, die auf Paketfilterung basieren, können fragmentierte Pakete gefährlich werden, weil je nach Anzahl der fragmentierten Pakete nur im ersten Fragment die TCP-Portnummer enthalten ist. Da nun fragmentierte Pakete ohne TCP-Port durchgelassen werden müssen, können z.B. zu kurze Pakete, bei denen die TCP-Portnummer erst im zweiten Paket erscheint, den Paketfilter vor große Probleme stellen. Es ist zum Beispiel denkbar, dass die Firewall häufige Verbindungswünsche, bei denen das SYN Flag gesetzt ist, nur IP Adressen akzeptiert, die vorher festgelegt wurden. Wenn sich jedoch nun zwei Fragmente eines SYN-Paketes überlappen, so lässt sich die Firewall überwinden und lässt die Pakete passieren. Mittels Fragmentierung lassen sich verschiedene Angriffe realisieren. Je nachdem wie gut oder schlecht der TCP Stack programmiert wurde, sinkt oder steigt die Anfälligkeit für Angriffe auf Basis fragmentierter Pakete. Einer der berühmtesten Angriffe ist der so genannte Ping of Death, bei dem statt der höchstmöglichen Anzahl von 65 535 Bytes das IP Paket eine noch höhere Anzahl von Bytes hat und somit den TCP Stack des betreffenden Systems

vor sehr große Probleme stellen kann.
Mit dem Befehl:

```
nemesis ip -S Quell-IP -D Ziel-IP -F {d,M,R} offset -P payload.txt
```

lassen sich fragmentierte Pakete erstellen und versenden. Heutige Linuxdistributionen enthalten einen über die Jahre gepflegten und aktualisierten TCP/IP Stack und sind gegen Fragmentierungsangriffe weitestgehend immun. Eine weitere Möglichkeit besteht im Blocken von speziellen und kaum vorkommenden IP-Fragmentoptionen an der Firewall. Darüber hinaus gibt es die Möglichkeit im Kernel, den Parameter ip_always_defrag zu aktivieren, sodass standardmäßig alle fragmentierten Pakete defragmentiert werden. Darauf wird im Abschnitt Verteidigungsmaßnahmen gegen Angriffe auf Protokollebene noch genauer eingegangen. [12]

3.1.3 ICMP-Angriffe

Das ICMP-Protokoll wurde ursprünglich dazu entwickelt, im Falle eines auftretenden Fehlers dem absendenden Host eine Nachricht über das Netzwerkproblem zuzustellen. Dieser absendende Host hat nun die entsprechende Aufgabe, Maßnahmen einzuleiten, um diese Netzwerkprobleme zu lösen. Oft ist es jedoch so, dass Router oder bestimmte Hosts diese ICMP-Nachrichten automatisch auswerten und entsprechende Aktionen durchführen. Danach führen sie meist selbst eine Rekonfiguration aus und übernehmen diese als Standard. Angreifer können dieses Verhalten nun missbrauchen und manipulierte und synthetisch erzeugte ICMP-Nachrichten erstellen. Sie sind somit in der Lage, Computersysteme zu ganz speziellen Aktionen zu bewegen. Die Ziele von ICMP-Angriffen sind die "Beeinträchtigung der Funktionalität des Netzwerkes" und die "Veränderung der Vermittlungspfade mit darauffolgendem Systemeinbruch". ICMP-Pakete werden mittels IP-Paketen verschickt und man kann sie nicht mit Protokollen wie z.B. TCP vergleichen, da sie ein Teil des Internet-Protokolls sind, der nicht deaktiviert werden kann. Als Sender von ICMP-Paketen treten Sendestationen auf, bei denen ein Fehler entdeckt wurde und empfangen wird das Paket vom Absender des Originalpaketes. Somit erfährt nur der Absender von dem Netzwerkproblem, nicht aber Zwischenstationen. Mittels eines Typenfeldes kann man die verschiedenen ICMP-Nachrichten klassifizieren. Die Wichtigsten hierbei sind Destination unreachable, source Quench und Redirect. Wenn nun ein Paket nicht an eine Adresse gesendet werden kann, so sendet der letzte Router die ICMP-Nachricht: destination unreachable. Ferner

werden neben dem eigentlichen Paketheader auch noch die ersten 64 Datenbytes des entsprechenden Paketes mit dem ICMP-Paket übertragen. Damit kann man der sendenden Station mitteilen, welche Internetverbindung abgebrochen ist. Oft besteht zwischen 2 Internetrechnern eine große Anzahl an Routern, die zu verschiedenen Internetadressen gehören, allerdings sind nicht alle ICMP-Implementationen fähig, die Port-Informationen des ICMP-Paketes zu analysieren und brechen somit die Verbindung ab. Somit kann ein Angreifer innerhalb kürzester Zeit eine Menge Internetrechner vom Netz abkoppeln. Des Weiteren kann man die Netzlast zwischen 2 Internetrechnern erhöhen, indem man ständig die ICMP-Nachricht "Fragmentation Needed and DF set" versendet. Darauf folgend müssen die durchleitenden Stationen ihre Daten fragmentieren und in kürzeren Paketen versenden. Dadurch wird die Netzlast erheblich erhöht. Weiterhin kann durch ein fehlerhaftes oder gewolltes Anwenden der ICMP-Nachricht "Source Quench" der Datenverkehr zwischen 2 Internetrechnern extrem gestört werden, da dies Router veranlasst, die Übertragunsrate der Verbindung stetig zu reduzieren, bis keine "Source Quench"-Nachrichten mehr ausgesendet werden. Mittels der ICMP-Nachricht "Redirect" kann man Router dazu veranlassen, eine abweichende Route zum Zielsystem für die Datenpakete zu verwenden. Dies kann wiederum missbräuchlich verwendet werden, um z.B. Datenstrom über einen eigenen Rechner umzuleiten. Router sollten jedoch immun gegen solche Angriffe sein und das Routing nur mittels der eigenen Routing-Tabellen durchführen. Ein weiterer Angriff mittels ICMP Nachrichten ist mit dem ICMP Echo Request möglich. Hierbei werden überlange, fragmentierte Pakete (aufgeteilte Pakete auf mehrere andere Teilpakete) an den Zielhost versendet, die am Zielsystem wieder zusammengesetzt werden und dabei einen Systemabsturz bewirken. Dieser, auf dem Echo Request Paket basierende Angriff, wird auch Ping of Death genannt. Gegen die beschriebenen Angriffe gibt es nur wenig Möglichkeiten, da ICMP-Nachrichten eine notwendige Komponente des Internet-Protokolls ist. Es ist jedoch möglich, Router so einzustellen, nur eine bestimmte Anzahl von ICMP-Nachrichten pro Zeitschlitz durchzuleiten. Normalerweise treten ICMP-Nachrichten nur sehr selten auf und moderne Überwachungssoftware sorgt dafür, dass bei übermäßiger Anzahl von ICMP-Paketen eine warnende Nachricht versendet wird. Der aktuelle Linux Kernel bietet jedoch einige Möglichkeiten, sich gegen ICMP-Angriffe zur Wehr zu setzen.

3.1.4 SYN-Flooding

SYN-Flooding wird ermöglicht durch eine Implementationsschwäche innerhalb des TCP/IP-Protokolls. Wenn eine TCP-Verbinung nur halb geöffnet ist, d. h. nur das erste SYN-Paket ge-

sendet wurde, behält der TCP/IP Stack diese Verbindung weiterhin im Speicher, um bei später eintreffenden Paketen die Verbindung noch korrekt aufzubauen. Sollte nun jedoch ein Angreifer eine große Menge an SYN-Paketen an das anzugreifende System senden, so kann der interne Speicher aufgebraucht werden und das System kann abstürzen. Es ist jedoch bei heutigen Implementationen des TCP-Stacks möglich, die so genannte Backlist Queue zu vergrößern, sodass der Dienst nicht gleich beim ersten Eintreffen von ein paar hundert SYN-Anfragen geschlossen wird, bzw. nicht mehr erreichbar ist. [12]
Mittels noch zu beschreibender Kernel Parameter kann man Linux sehr gut gegen SYN-Flooding Angriffe schützen. Zu Testzwecken kann man mit Hilfe des Programms nemesis_syn_flood.pl einen SYN-Flood Angriff nachstellen. Mit dem Tool anti_syn_flood.pl kann man solchen SYN-Flooding Angriffe wirkungsvoll bekämpfen. Auf das Programm anti_syn_flood.pl wird im weiteren Verlauf noch detaillierter eingegangen.

3.1.5 Man-in-the-Middle-Angriffe

Man-in-the-Middle-Angriffe werden meist in Computernetzwerken durchgeführt, um den Datenverkehr von zwei Computersystemen über den Rechner des Angreifers zu routen, sodass dieser die Kommunkiationsbeziehung überwachen und verändern kann. Dazu muss er mindestens logisch zwischen diese beiden Computer geschaltet sein, ansonsten muss sich der Angreifer physisch zwischen den beiden Rechnern befinden, was meist nicht der Fall ist. Um sich optimal zwischen zwei Computersystemen zu plazieren, muss der Angreifer die Kontrolle über einen Router haben, durch den der Datenverkehr geschleust wird. Oft ist dies ein nicht bekannter Internetrouter, sondern der Rechner des Angreifers, der dann als Router fungiert und den Verkehr zwischen beiden kontrolliert und weiterleitet. Um solch einen Angriff erfolgreich durchzuführen, werden oft ARP-Spoofing und ARP-Cache-Poisoning verwendet, um Kontrolle über den Datenverkehr zu erlangen. Mit dem bereits beschriebenen Tool Ettercap kann man verschiedene Man-in-the-Middle Angriffe durchführen. Am besten lassen sich solche Angriffe verhindern, wenn der Datenverkehr verschlüsselt wird, wie es mit z.B. Secure Shell der Fall ist. Darüber hinaus müssen allerdings die Fingerpints von SSH über ein zuverlässiges Medium ausgetauscht werden, weil sonst wieder Gefahr für einen Angriff besteht. [13]

3.2 Verteidigungsmaßnahmen gegen Angriffe auf Protokollebene

Die hier besprochenen Vorkehrungen gegen eben genannte Angriffe sind leicht umzusetzen und erfordern es nicht, dass ein eigener Kernel übersetzt wird. Diese Verteidigungsmaßnahmen sind einfach umzusetzen und sollten entsprechend am eigenen System durchgeführt werden.

Kernelhärten gegen IP- und ICMP-basierende Angriffe

Im Abschnitt 6 wird im Rahmen der Verteidigungsmöglichkeiten gegen TCP-, IP- und UDP-Angriffe darauf hingewiesen, wie man den eigenen Kernel modifizieren kann, um Schutz gegen diese Angriffe zu bieten. Postiv hierbei ist, dass das System nicht neu gebootet und auch kein Kernelpatch eingespielt werden muss, um die Schutzfunktionen zu aktivieren. Da diese Vorgehensweise noch beschrieben wird, soll hier auf Kapitel 6 verwiesen werden.

Programm anti_syn_flood.pl gegen SYN-Flooding Angriffe

Das Perl Programm anti_syn_flood.pl verwendet das Perl Module Net::RawIP zum Assemblieren und Disassemblieren von IP-, TCP- und UDP-Paketen. Es loggt die Anzahl der eintreffenden SYN-Pakete mit. Im Falle eines SYN-Flood-Angriffes schickt das Programm für jedes eintreffende und falsche SYN-Paket ein TCP-Paket mit gesetztem FIN-Flag und beendet somit die Verbindung, die sonst noch in der Backlog-Queue gelistet worden wäre. Damit verhindert das Programm, dass die Backlog-Queue verstofft und der Dienst nicht mehr erreichbar ist.

3.3 Angriffe auf Anwendungsebene

Fehlerhafte Implementationen können dazu führen, dass man mit Hilfe von Buffer Overflows, Heap oder Stack Overflows ein System kompromittieren und höhere Systemrechte erlangen kann. Im weiteren Verlauf wird detaillierter auf das Thema Buffer Overflows eingegangen, die Themen Heap und Stack Overflows sind nicht Bestandteil dieser Arbeit.

3.3.1 Mangelnder Syntaxcheck

Wenn Buffer Overflows auftreten und diese aufgrund mangelnder "Kontrolle der Länge von externen Eingaben" in Erscheinung treten, so kann dies ausgenutzt werden, wenn die Syntax von eingegebenen Daten unzureichend gesichert ist. [12] So genannte Parser prüfen die Eingabe der Daten und zerlegen diese dann in Befehle, Parameter und Daten. Wenn nun jedoch diese Parser schlecht und fehlerhaft programmiert sind, können Angriffe auf diese Systeme gefahren werden. Besonders bei der Behandlung von Sonderzeichen treten viele Fehler aufgrund falschem Parsings auf. Wenn nach dem Parsen noch Sonderzeichen im Datenstrom vorhanden sind, so kann die anschließende Bearbeitung durch die Applikation oder das Betriebssystem zu katastrophalen Konsequenzen führen. Bei älteren Apache Versionen (der Open Source Webserver) kam es durch unzureichende Bearbeitung der Zeichen "%A" (Newline) und "%20" (Space) zu Fehlern bei der Ausführung von CGI-Programmen. So war es z.B. möglich, sich die Passwortdatei des Systems anzugucken. Jedoch gab und gibt es teilweise auch heute noch Probleme bei der Verarbeitung von "..\", "../", "|" und ";". Webserver, bei denen diese Fehler wiederholt auftreten, erlauben es einem Angreifer z.B. das virtuelle Rootverzeichnis des Webservers zu verlassen und sich auf der Festplatte des Rechnersystems umzusehen. Wenn der Webserver nun noch mit Root-Rechten arbeitet , diese vorher nicht durch Privilegdropping vermindert hat, kann sich der Angreifer die verschlüsselte Shadow Datei herunterladen und mit Hilfe von bekannten und oben beschriebenen Crackprogrammen versuchen, Benutzernamen und zugehörige Passwörter zu erraten. Problematisch sind auch CGI-Programme, die eingegebene Daten schlecht oder sogar fehlerhaft parsen und somit zu Sicherheitslücken auf dem System beitragen können.

3.3.2 Race Conditions

Bei den so genannten Race Conditions findet ein Wettlauf zwischen zwei Anwendungen statt. Hierbei nutzt das eine Programm bestimmte Systemressourcen, wie Dateien, Speicherbereiche oder Interrupts, sichert diese Daten jedoch nicht oder kaum gegen einen Missbrauch von außen ab. Das Progamm des Angreifers kann nun auf diese speziellen Ressourcen in unberechtigter Weise zugreifen und diese manipulieren, was bei der urpsrünglichen Anwendung zu Seiteneffekten führen kann. Der Zugriff seitens des angreifenden Programms muss allerdings in genau dem richtigen Moment stattfinden, damit der Angriff gelingt. Oft finden Angriffe mittels Race Condition auf die temporären Daten einer Anwendung statt, die entsprechend geändert werden und somit zu einem für

den Angreifer positiven Nebeneffekt bei der originalen Anwendung führen. [15]

3.3.3 Portscanning mit Nmap als Angriffsvorbereitung

Der Portscanner Nmap wurde von dem Hacker Fyodor programmiert. Nmap unterstützt verschiedene Scantechniken. Auf die Wichtigsten soll hier kurz eingegangen werden, um einen Rechner auf offene Ports zu testen. Die einfachste Scantechnik wird mit dem Flag "Nmap -sT HOST" eingeleitet und kennzeichnet den TCP connect()-Scan. Hierbei wird der connect()-System-Call des Betriebssystems genutzt, um eine Verbindung zum Port der entfernten Maschine aufzubauen. Stellt sich heraus, dass sich der entfernte Port im Zustand "LISTENING" (dt. abhörend) befindet, so gilt der connect()-System-Call als erfolgreich ausgeführt und der betreffende Port ist offen und wird von Nmap entsprechend gekennzeichnet. Um so einen connect()-System-Call auszuführen, bedarf es keiner weiteren Rechte, wie z.B. Root-Rechte, sodass dieser TCP connect()-Scan von jedem unprivilegierten Benutzer ausgeführt werden darf. Allerdings kann man diese Art des Scannens sehr leicht erkennen und entsprechend protokollieren. Mit Hilfe der Option "-sS" kann man den so genannten TCP SYN-Scan initialisieren und dann entsprechend durchführen lassen. Diese Art des Scannens wird auch als "halb-offener" Scan bezeichnet, da hier keine vollständige TCP-Verbindung nach dem 3-Way-TCP-Handshake aufgebaut wird. Um diesen Scan durchzuführen, schickt Nmap ein TCP-Paket mit gesetztem SYN-Flag an den zu scannenden Host. Wenn der Host nach Richtlinie des 3-Way-TCP-Handshake handelt, sendet er auf diese SYN-Anfrage natürlich ein Paket mit gesetztem Flag SYN-ACK und kennzeichnet damit, dass sich der Port im Zustand LISTENING befindet und Nmap diesen Port anschließend als offen kennzeichnet. Sollte der gescannte Host jedoch ein TCP-Paket mit gesetztem RST-Flag auf diesen Scan antworten, so ist der Port geschlossen. Diese Scantechnik ist standardmäßige Methode zum Aufspüren von offenen Ports für privilegierte Benutzer, da hier die "Generierung verhältnismäßig exotischer Paket-Sequenzen von Noeten ist". Beim TCP Stealth FIN-Scan verschickt Nmap TCP-Pakete mit gesetztem FIN-Flag an die entsprechenden Ports der entfernten Maschine und wartet auf die Antwort auf dieses TCP-Paket. Sollte der Port geschlossen sein, so antwortet er mit einem TCP-Paket mit gesetztem RST-Flag. Wenn er offen ist, wird die Anfrage einfach ignoriert. Im Gegensatz zum TCP Stealth FIN-Scanning werden beim TCP Stealth XMAS-Scan TCP-Pakete mit gesetzten FIN, URG und PSH Flags generiert und verschickt. Ein geschlossener Port antwortet auch hier mit einem TCP-Paket mit gesetztem RST-Flag, offene Ports ignorieren diese Anfrage. Beim TCP Stealth NULL-Scan werden TCP-Pakete generiert, bei denen kein Optionsflag gesetzt ist und an den

Port des entfernten Hosts gesendet wird. Da man weiß, dass offene Ports solch eine Anfrage ignorieren, kann man leicht erkennen, wann ein Port offen oder geschlossen ist. Sollte es der Fall sein, dass der Port geschlossen ist, so antwortet er hier auch wieder mit einem TCP-Paket mit gesetztem RST-Flag. Beim UDP-Scan wird ein 0 Byte UDP-Datagramm an den entfernten Port des Rechners gesendet. Sollte der Port geschlossen sein, so reagiert das entfernte System auf diesen Scan, indem es eine ICMP port unreachable Nachricht zurücksendet und Nmap daraus auf einen geschlossenen Port schlussfolgern kann. Sollte diese ICMP Nachricht ausbleiben, so ist der Port offen. Der Version-Scan kann eingesetzt werden, wenn man wissen möchte, welche Softwareversion auf dem entfernten Rechner an den entsprechenen Ports genutzt wird. Um zu testen, welcher Host im Netzwerk aktiv und erreichbar ist, kann man den Ping-Scan einsetzen. Nmap sendet hierzu eine ICMP echo request-Anfrage an die zu untersuchende IP des Rechners und wartet anschließend auf eine Antwort. Der entfernte Host antwortet im postiven Fall mit einer ICMP echo reply Nachricht und daraus schließt Nmap dann, dass der Host aktiv und erreichbar ist. Eine Firewall kann den einen Portscan abblocken. Darüber hinaus gibt es die Möglichkeit, den Linuxkernel zu patchen, sodass er auf TCP Pakete mit gesetztem RST-Flag nicht antwortet. Somit verlieren FIN, XMAS und NULL-Scans ihre Daseinsberechtigung. Diese Patchmöglichkeit wird im weiteren Verlauf noch erläutert. [16]

3.3.4 Password Sniffing

Unter dem Begriff Password Sniffing versteht man das Mitlesen bzw. das Mitschneiden von Login-Informationen aus einem bestehenden Netzwerkdatenstrom. Hierbei wartet die betreffende Anwendung im Hintergrund und durchsucht den ankommenden Datenstrom nach entsprechenden Daten und zeichnet diese entsprechend mit. Unter Linux ist das Programm Ethereal ein so genannter Protokoll-Analyser, der den Datenverkehr von mehreren hundert Protokollen mitschneiden kann. Somit ist es möglich, Passwörter mit aufzuzeichnen, wenn die Kommunikationsbeziehungen zwischen zwei Rechnern ohne Verschlüsselung erfolgt. Mit Hilfe der Programmsuite Dsniff kann man hervorragend Passwörter mitschneiden. Verschlüsselte Verbindungen beim Abholen von Nachrichten oder ein kompletter Verzicht auf unverschlüsselte Protokolle können hier ein Einfallstor schließen. [12]

Mit Hilfe des Secure Shell Protokolles ist es möglich, verschlüsselte Datenübertragung und Datenkommunikation zu realisieren. Es ist in der Lage, das unverschlüsselte File Transfer- und Telnet Protokoll zu ersetzen und somit Password Sniffing gegen diese verschlüsselten Protokolle

fast unmöglich zu machen.

3.3.5 Password Guessing

Unter dem Begriff "Password Guessing" versteht man das einfache Erraten von Passwörtern. Dabei werden bekannte Logins wie administrator, guest, service, field oder root genommen und nun durch Ausprobieren versucht, das entsprechende Passwort für diese Zugangskennung zu erraten.
Auch sind Standardlogins wie guest/guest oder admin/admin in der Regel Einfallstore, die auch von den unerfahrensten Angreifern ausgenutzt werden. Im Anhang befindet sich das entworfene Perlprogramm genpasswd.pl, womit man sichere Passwörter erstellen kann, die 10-11stellig sind und den oben beschriebenen Anforderungen entsprechen. Es ist sehr einfach aufgebaut und nimmt aus einem vorgegebenen Zeichenvorrat Buchstaben,Ziffern und Sonderzeichen heraus und generiert dann ensprechend sichere Passörter. Mit dem zugehörigen Programm support_genpasswd.pl ist es auch möglich, auf einmal mehrere Passwörter zu erstellen. Will man beispielsweise 20 Passwörter erstellen lassen, so ruft man das Programm einfach mit "perl support_genpasswd.pl 20" auf und dann werden ensprechende 20 Passwörter erstellt. Mittels einfacher Programme, wie oben beschrieben, ist es heutzutage möglich, sichere Passwörter zu erstellen. Diese sollten nach den beschriebenen Richtlinien erstellt worden sein, damit eine Sicherheit gegenüber Password Cracking oder Password Guesssing gegeben ist. Weiterhin wird sehr empfohlen, die eigenen Authentifizierungsmaßnahmen entsprechend folgendem Hinweis zu ändern. Standardmäßig werden die Passwörter in der Shadow Datei heutzutage mit dem veralteten DES (Digital Encryption Standard) verschlüsselt. Auf aktuellen Pentium4- und Athlon 64- Systemen ist es mit Hilfe von John the Ripper möglich, über 200 000 Passwortkombinationen pro Sekunde auszutesten. Wenn man nun die Verschlüsselung von DES auf OpenBSD Blowfish umstellt, kann man mit aktuellen Einzelrechnersystemen nur noch knapp 3000- 4000 Passwörter pro Sekunde erraten, was einer enormen Sicherheitssteigerung gegenüber DES entspricht, da es nun mit Hilfe von John the Ripper viel länger dauert, entsprechende Passwörter zu knacken bzw sie zu erraten. [12]

3.3.6 Ausnutzung von Buffer Overflows

Beschreibung und Definitionen von Buffer Overflows

Der aus dem Englischen stammende Begriff "buffer overflow" bedeutet Pufferüberlauf und ist eine

der am häufigsten auftretenden Sicherheitslücken in heutigen Programmen. Diese Sicherheitslücken lassen sich je nach Programm lokal oder remote ausnutzen, um unrechtmäßigen Zugriff auf ein System oder Programm zu erlangen. Buffer Overflows funktionieren nach folgendem Prinzip. In einem Programm werden zu große Datenmengen in einen Speicherbereich geschrieben, der dafür unterdimensioniert ist. Dadurch können nachfolgende Informationen im Speicher verändert und überschrieben werden. Durch diesen Vorgang kann die Rücksprungadresse eines Unterprogramms mit Daten überschrieben werden und somit fremder Code eingeschleust werden, der zum Beispiel einem Angreifer eine Shell öffnet. Auf Linuxrechnern ist der Rootzugriff das höchste zu erlangende Privileg bei der Nutzerverwaltung. Er ist also ein begehrtes Ziel für viele Buffer Overflow Angriffe. Buffer Overflow Attacken sind aktuelle Themen im Bereich Rechnersicherheit, denn sie können nicht nur über Netzwerke sondern auch lokal angewendet werden. Oft kann man sich gegen Buffer Overflows nur schützen, indem man so genannte Fehlerkorrekturprogamme (Patch) auf dem betreffenden System einspielt. Diese werden entweder von den Herstellern produziert oder von unabhängigen Sicherheitsexperten im Internet veröffentlicht. Buffer Overflows werden oft von Internetwürmern wie Nimda oder CodeRed ausgenutzt und tragen so zur raschen Verbreitung dieser Schädlinge bei. Das Problem der Pufferüberläufe entsteht zumeist durch unnachsichtige Programmierung, aber gerade die Von-Neumann-Architektur aktueller Rechnersysteme trägt zu diesem Problem bei. Hier lagern Daten und das eigentliche Programm im gleichen Speicher. Jedoch treten Buffer Overflows nur bei assemblierten oder compilierten Programmiersprachen auf. Interpretersprachen wie Perl sind, abgesehen vom eigentlichen Interpreter, nicht für solche Probleme anfällig, da die Speicherbereiche für Daten ständig durch den Interpreterprozess kontrolliert werden.

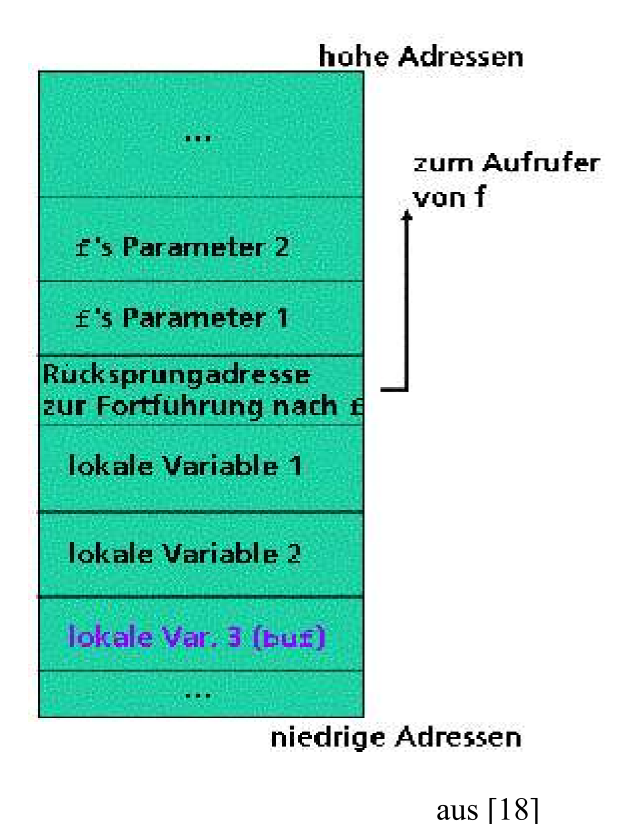
Funktionsweise von Buffer Overflows

Mittels der Programmiersprache C ist es möglich, direkt auf den Hauptspeicher des Betriebssystems zuzugreifen. Dies geschieht mittels der Verwendung von Pointern und Arrays. Hierbei muss seitens des Programmierers selbst darauf geachtet werden, dass an den angesprochenen Speicherzellen tatsächlich Daten stehen dürfen. Besonders die Länge eines Datenwertes in Bytes muss der Länge des vorher festgelegten Speicherbereiches entsprechen. Gerade bei der Programmierung machen Leute den Fehler, dass Speicher alloziert und dann mit einem größeren tatsächlichen Wert überschrieben werden. Ein einfaches Beispiel soll anhand einer kleinen C-Funktion deutlich machen, wie es zu solchen Pufferüberläufen kommen kann. Die folgende Funktion verursacht einen sehr offensichtlichen Pufferüberlauf, wenn sie mit einem Parameter aufgerufen wird, der größer als

12 Bytes ist. Das Array buff ist 12 Bytes groß und so führt ein Aufrufen mit einem Parameter der länger als 12 Bytes ist zu einem Buffer Overflow:

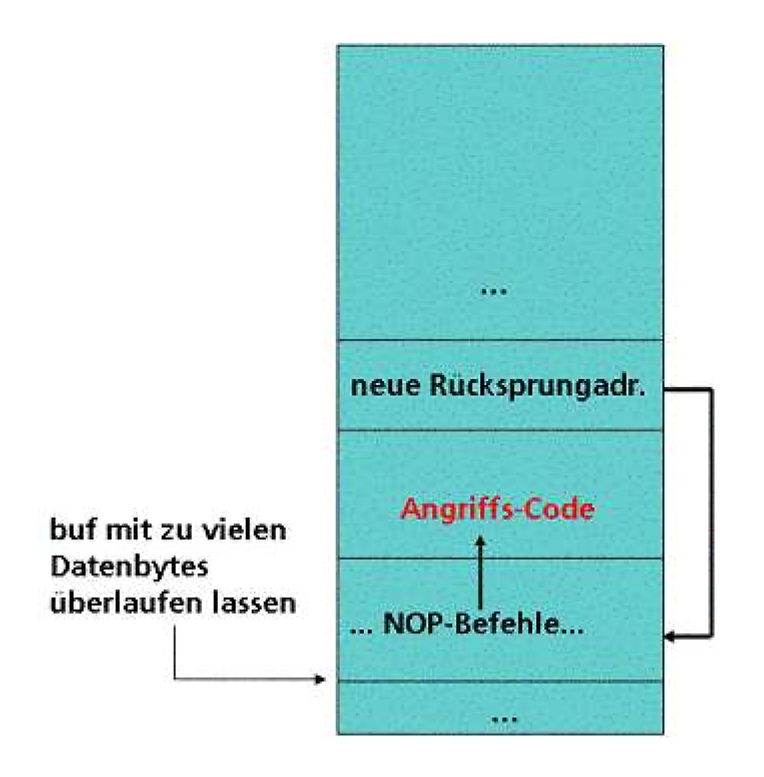
```
void f(char* in) {
    char buf[12];
    strcpy(buf, in);
}
```

Die eben angesprochene Problematik tritt in gleicher Art und Weise auch bei C++-Programmen auf, da auch hier Arrays und Pointer wie in C verwendet werden. Bei Programmiersprachen, die keinen direkten Zugriff auf den Speicher erlauben, wie Java, Visual Basic und Interpretersprachen wie AWK und Perl treten solche Probleme nicht auf, da hier die Laufzeitumgebung der Programmiersprache die Speicherzugriffe kontrolliert. Das oben erwähnte Programmfragment führt nur zu einem Programmabsturz, jedoch kann man Buffer Overflow behaftete Programme im günstigsten Fall auch so ausnutzen, dass er eine Rootshell und somit unberechtigte Privilegien vom System erhält. Dies tritt meist auf, wenn ein Programm Daten von einer Netzwerkverbindung liest oder wenn dem Programm Daten zur Verarbeitung übergeben werden. Falls nun der angegriffene Buffer auf dem Laufzeitstack liegt, kann man den Puffer mit eigenen eingeschleusten Daten überlaufen lassen und somit die auf dem Laufzeitstack liegende Rücksprungadresse, die hinter dem Puffer liegt, gezielt überschreiben, sodass das Programm bei der Rückkehr aus dem Unterprogramm an einer Stelle fortgesetzt wird, die der Angreifer vorher ausgesucht hat. Diese Stelle wird meistens nicht im originalen Code zu finden sein, sondern in einem eigens eingeschleusten Angriffscode, oft auch Shellcode genannt. Die oben vorgestellte C-Funktion ist z.B. für solch einen Buffer Overflow Angriff geeignet. Die folgenden Grafiken zeigen einen solchen Angriff auf einen Buffer innerhalb des Laufzeitstack der Funktion f().[18]



aus [18]

Abbildung 3.1: Stack nach dem Aufruf von f und vor Pufferüberlauf



aus [18]

Abbildung 3.2: Stack nach dem Pufferüberlauf von buf

3.3.7 Schadsoftware: Würmer, Trojaner und Rootkits

Ein Computerwurm ist ein Schädlingsprogramm, das so ähnlich wie Viren, Dateien und Programme infiziert. Im Allgemeinen verbreiten sich heutige Computerviren über das Internet als email-Anhängsel oder sie nutzen Schwachstellen im Betriebssystem aus. Ein Wurm kann eine Schadensroutine haben, muss es aber nicht. Wie der Computervirus verbraucht der Wurm Ressourcen auf dem befallenen System, um sich weiter zu verbreiten, sodass dies immensen wirtschaftlichen Schaden verursachen kann (z.B. Rechner an der Börse sind nicht mehr zu erreichen). Unter Windows bekommen Würmer oft doppelte Dateinnamen wie scooter_one.mp3.vbs und Windows zeigt oft nur die erste Dateiendung an. So wird der Anwender zum Klicken auf die Datei animiert und der Wurm kann starten und sich seine Ziele im Internet aussuchen. Windowswürmer schreiben sich oft in die Registry oder in den Autostartordner ein, um bei jedem Rechnerstart ebenfalls aktiv werden zu können. [19]
Trojanische Pferde sind Programme, die neben der Programmfunktion auch eine schädliche Routine eingebaut haben. Oft verfügen Trojanische Pferde über eine nützliche Funktion, die schädliche Funktion läuft im Hintergrund ab, ohne dass dies vom Anwender bemerkt wird. Es gibt verschiedene Varianten von Trojanern. Es gibt Trojanische Pferde, die aus einer einfachen ausführbaren Datei bestehen und außer der schädlichen Funktion keinerlei nützliche Funktion haben. Beim Starten des Programms scheint es, als ob sich das Programm gleich wieder beendet, im Hintergrund wird jedoch der Trojaner gestartet. Meist wird die Originaldatei gelöscht, da der Anwender keinen Nutzen von dem Programm hat, im Hintergrund arbeitet jedoch der aktive Trojaner. Des Weiteren gibt es Trojanische Pferde, die im Vordergrund ein nützliches Programm anbieten (z.B. Stand-Alone Virenscanner) und dann im Hinterhalt ein schädliches Programmfragment starten. Eines haben Trojaner gemeinsam, meistens installieren sie sich tief im Systemkern und werden so bei jedem Neustart des Systems mitgeladen. [20]
Rootkits sind kleine Programme, ähnlich den Trojanern, die man einsetzt, nachdem man in ein Computersystem eingedrungen ist. Sie dienen meistens dem Zweck, einfachen Zutritt zum System zu gelangen, ohne sich lange mit den Sicherheitsprozessen für das betreffende System auseinanderzusetzen. Es gibt sie für verschiedene Betriebssysteme wie Microsoft Windows XP oder die diversen Linuxderivate. Rootkit bedeutet meist eine Ansammlung von Betriebssystemtools, wie sie das betreffende System meist schon mitbringen. Unter Linux werden die Programme "ps", "nestat","w" oder auch das "passwd"-Programm gerne gegen Crackertools ausgetauscht, sodass diese dann nicht mehr die verräterischen Spuren des Crackers für den Systemadministrator

aufzeigen. Dies alles dient dem Ziel "root"-Rechte auf dem betreffenden System zu sichern. Typischerweise versteckt ein Root-kit login-Prozesse, verändert Logs und beinhaltet Software, um den Datenfluss zwischen Terminals, Netzwerksessions oder der Tastatur zu überwachen. Meistens helfen sie dem Angreifer, einfachen Systemzugriff zu erlangen, z.B. kann ein Rootkit jedesmal, wenn man sich auf einen speziellen Port hin verbindet, eine Shell erstellen, mit der man dann am System arbeiten kann, ohne vorher ein Passwort eingegeben zu haben.
Es gibt 2 Arten von Rootkits. Kernel- und anwendungsbezogene Rootkits. Kernelrootkits sind meist sehr raffiniert geschriebene Programmcodes, die mit originalem Kernelcode ersetzt werden sollen, um die Angriffspuren eines Crackers zu verwischen. Hierfür werden oft Kernelmodule geschrieben und dann in den Kernelbereich des Speichers geschrieben. Dem Kernelbereich des Speichers wird generell eine höhere Priorität zugewiesen als Programmen, die im so genannten Benutzermodus laufen. Somit können Rootkits, die in den Kernelspeicher geladen werden, Überblick über alle Systemprozesse behalten. Oft ersetzen sie Systemaufrufe mit neuen Varianten bestehender Systemaufrufe, um weiterhin ungesehen auf dem System zu bleiben. Ein Deaktivieren von Kernelmodulen bei der Kernelkompilierung kann die Gefahr von Kernelrootkits minimieren. Anwendungsbezogene Rootkits sind meist reguläre Anwendungen, die durch gefakte, trojanische Varianten ausgetauscht werden. [21]

3.4 Verteidigungsmaßnahmen gegen Angriffe auf Anwendungsebene

Korrektur des Quellcodes durch Dritte

Mangelndem Syntaxcheck kann man eigentlich nur vorbeugen, wenn man vor der Programmierung Sicherheitsrichtlinien aufstellt und darin z.B. deklariert, dass sicherheitsrelevante Codeabschnitte von mehreren Programmierern kontrolliert werden müssen, sodass kein kritischer Fehler in der Applikation bleibt. Vorbeugende Maßnahmen gegen Race Conditions sind vor allem die überlegte und sichere Programmierung von Anwendungen, bei denen der Zugriff auf benutzte Systemressourcen kontrolliert und gegebenfalls geschützt werden kann. So ist es unter Linux mittels so genannter Flock-Mechanismen möglich, den Zugriff für andere Programme auf z.B. eine Datei zu sperren, bis die Anwendung mit dem Schreiben und Schließen der Datei fertig geworden ist.

Abblocken von Portscans

Im Linuxkernel findet man im Verzeichnis "net/ipv4" die Datei "tcp_ipv4.c". Dort sucht man ab Zeile 1154 nach folgendem Codeabschnitt :

```
static void tcp_v4_send_reset(struct sk_buff *skb)
{
        struct tcphdr *th = skb->h.th;
        struct tcphdr rth;
        struct ip_reply_arg arg;

        /* Never send a reset in response to a reset. */
        if (th->rst)
                return;
        ...
}
```

Dieses Fragment ändert man dahingehend, dass der return-Befehl eingefügt wird, sodass kein RST-Paket mehr gesendet wird. Nach der anschließenden Neukompilierung des Kernels sendet dieser keine TCP-Pakete mit RST-Flag mehr und macht somit Portscans auf Basis von FIN-, XMAS- und NULL-Flags bei TCP-Paketen wirkungslos. Wenn man nun mit den genannten Methoden gescannt wird, so zeigt Nmap keine offenen Ports auf dem eigenen Server an. [24]
Der oben genannte Code wird geändert:

```
static void tcp_v4_send_reset(struct sk_buff *skb)
{
        struct tcphdr *th = skb->h.th;
        struct tcphdr rth;
        struct ip_reply_arg arg;

        return; // Änderung, dass nie ein RST-Paket gesendet wird
                // Verhinderung der FIN, XMAS und NULL Portscans

        /* Never send a reset in response to a reset. */
        if (th->rst)
```

```
                return;
        ...
}
.
```

Weiterhin besteht die Möglichkeit, den Linuxkernel mit dem Security Patch Grsecurity zu patchen und IPTables mit dem Stealth Patch zu modifizieren, sodass ebenfalls Portscanns abgeblockt werden.

Richtlinien für sichere Passwörter

Um ein Password Guessing oder Password Cracking zu erschweren, kann man folgende vier Eigenschaften für ein Passwort geltend machen. Sie sollten schwierig zu erraten sein, nicht aus einem Wörterbuch stammen, trotzdem leicht zu merken sein, regelmäßig geändert werden und außerdem noch gut geschützt und verschlüsselt aufbewahrt werden. Des Weiteren sollten folgende Richtlinien bei der Erstellung von Passwörtern eingehalten werden, damit eine optimale Passwortsicherheit gewährleistet ist. Passwörter sollten möglichst alle 3 Monate geändert werden, sowie Groß- und Kleinbuchstaben, Ziffern und Sonderzeichen enthalten. Weiterhin kann man mit Hilfe einfacher Abkürzungen wie "I need a very short Password 4 my Account": "InavsP4mA" bereits gute Passwörter erstellen. Durch Wortkombinationen kann man die Sicherheit weiter erhöhen. Es bleibt zu erwähnen, dass Passwörter mindestens 8 - besser jedoch 10 Stellen haben sollten.
Mit den im Anhang gelisteten Programmen genpasswd.pl und support_genpasswd.pl kann man sichere Passwörter erstellen.

Maßnahmen gegen Buffer Overflows

Sichere Programmierung

Heutzutage wird die Programmierung von Software oft Firmen übergeben, die dann unter Zeitdruck die geforderten Aufgaben bewältigen müssen. Hierbei ist der Fakt, dass man Programmierfehler möglichst vermeiden soll, ein sehr schwieriges Unterfangen. Jedoch liegen auch Probleme in der Konzeption der Programmiersprache C, da C nicht automatisch Arrays und Pointer überprüft. Allerdings stellt sich die Verwendung von Standardfunktionen aus der Linux Standardbibliothek als problematisch dar. Für Funktionen wie strcpy() oder strcat() sollte man versuchen, entsprechenden

Ersatz zu nehmen. Eine Lösung für das Aufspüren von Fehlern im Quellcode stellen die so genannten Source Code Security Analyser dar. Die Firma Reliable Software Technologies stellt hierfür das Programm ITS4 zur Verfügung. Im Bereich der Open Source Programme gibt es auch das Programm Slint der Hackergruppe L0pht. Diese Analyseprogramme finden kritische Programmanweisungen und bieten auch gleich eine Beschreibung bzw. einen Verbesserungs-vorschlag für den Source Code an. [14]

Openwall Security Patch - non-executable stack area

Weitere Möglichkeiten zur Vermeidung von Buffer Overflows liegen in dem Versuch, dem potentiellen Cracker die Ausführung des zusätzlich eingefügten Maschinencodes zu verbieten. Wenn dies gelingt, dann kann der Buffer zwar weiterhin überlaufen, jedoch kann dann kein schädlicher Code mehr ausgeführt werden. Die meisten Buffer Overflow Exploits basieren auf der Überschreibung der Returnadresse einer Funktion auf dem Laufzeitstack. Wenn nun der Stack nicht ausführbar ist, ist es fast unmöglich, auf herkömmlichem Weg das Exploit zu nutzen. Hier greift der Openwallpatch, der genau dieses Feature dem Linuxkernel hinzufügt. Außerdem kann man Buffer Overflows ausnutzen, indem man die Returnadresse einer Funktion auf die Standardbibliothek libc verweisen läßt. Meistens ist das die Funktion system() und daran anschließend wird der Exploit ausgeführt. Jedoch sollte man bei Bekanntwerden eines Buffer Overflows in einem Programm immer die neuesten Patches und Updates einspielen, um die Sicherheit des Systems zu gewähren. [22]

Array-Grenzenüberwachung

Das Problem bei Buffer Overflows ist, dass der Buffer, beziehungsweise das entsprechende Array, mit zuviel Daten gefüllt wird, sodass diese dann überlaufen. Nun kann man durch die Überwachung des Arrays und seine Arraygrenzen sicherstellen, dass es nicht mehr zum gefürchteten Buffer Over-flow kommt. Somit ließen sich Pufferüberläufe geschickt verhindern. Jedoch ist hierbei jede Lese-, bzw. Schreibaktion auf ein Array zu kontrollieren, was sehr viel Systemressourcen aufbrauchen kann. Richard Jones und Paul Kelly haben einen Patch für die Kompilersuite gcc entwickelt, der genau diese Kriterien erfüllt. Jedoch trat bei praktischen Tests eine bis zu 30 fach verlangsamte Systemperformance auf, sodass dieser Patch bei aktuellen Linuxdistributionen kaum eingestetzt wird. [14]

Immunix StackGuard Kompiler

Eine weitere Möglichkeit, Buffer Overflows wirkungsvoll zu vereiteln, besteht in der Verwendung des StackGuard Compilers von Immunix. Dieser Kompiler ersetzt den Standardkompiler und er versucht, Pufferüberläufe zu verhindern, indem er an markanten Stellen innerhalb des Programmcodes Kontrollwerte, so genannte canaries, setzt. Wenn nun beim Rücksprung einer Funktion der canaries-Wert nicht mit dem original Gespeicherten übereinstimmt, wird das Programm beendet. Um die Sicherheit für bestehende Programme zu übernehmen, muss man diese nur mit dem StackGuard Kompiler übersetzten. Die wichtigste StackGuard Version ist stout. Diese Version hat die Funktion _canary_death_handler() implementiert, die immer dann einspringt, wenn ein Buffer Overflow entdeckt wird. Sollte dies eintreten, wird das Programm durch diese Funktion erfolgreich beendet und der Buffer Overflow Exploit verpufft wirkungslos. [14]

Libsafe

Die Libsafe ist eine speziell programmierte, dynamisch ladbare Linuxsystembibliothek, die potentiell gefährliche Funktionsaufrufe durch nicht Anfällige ersetzt. Dazu fängt Libsafe alle Aufrufe von gefährlichen Funktionen ab und ersetzt diese durch eigene, nicht Buffer Overflow Anfällige. Im Gegensatz zur Methode, bei der man den StackGuard Kompiler einsetzte, ist diese Art der Verhinderung von Pufferüberlauf-Exploits völlig Source Code unabhängig und kostet auch nicht so viel Performance. Einige wichtige Funktionsaufrufe, die ersetzt werden, sind strcpy(), getwd(), gets() oder realpath(). [14]

Schutz vor Schadsoftware

Das Programm F-Prot kann man zum Aufspüren und Löschen von Viren und Trojanern verwenden. Im Gegensatz dazu ist das Programm chkrootkit Bestandteil jeder aktuellen Linuxdistribution und es ist ein Systemprogramm, d.h. es kann nur mit Rootrechten gestartet werden. Nach dem Aufruf fängt das Programm automatisch an, Rootkits aufzusuchen und dann zu löschen. Am Ende gibt es eine kurze Zusammenfassung und stellt kurz dar, ob und wenn etwas gefunden wurde.
Das Programm Rootkit Hunter kann von rootkit.nl kostenlos bezogen werden. Es akzeptiert eine Menge Parameter und scannt nach mehr als 20 Rootkits. Die ständige Weiterentwicklung seitens des Authors Michael Boelen sichert Aktualität und somit bei ständiger Benutzung auch

ein relativ sicheres Linuxsystem.

Das Programm debian_patch_download.pl

Das Programm debian_patch_download.pl ist für Administratoren gedacht, die ein Linuxsystem auf Basis von Debian warten und pflegen müssen. Mit diesem Programm kann man nach aktuellen Fehlerkorrekturen suchen und diese dann auch gleich herunterladen lassen, um sie anschließend zu installieren.

3.5 Denial-of-Service-Angriffe

Neben den bekannten Spoofing-Angriffen zur Vortäuschung von falschen Paketen gibt es auch so genannte Denial-of-Service-Angriffe, deren Ziel es ist, einen bestimmten Dienst durch Überschwemmung mit Pakten unbrauchbar zu machen.

Broadcast-Angriffe

Eine weitere Möglichkeit zur Auslastung der eigenen Netzressourcen sind so genannte Broadcast-Angriffe. Das ICMP-Protokoll wurde u.a. dazu entwickelt, Statusmeldungen über die Erreichbarkeit von Internetrechnern zu versenden. Dazu dient der ICMP-Echo-Request, der versendet wird, um zu testen, ob der entfernte Rechner ansprechbar ist und auf Serviceanfragen reagieren kann. Ist dieser Rechner verfügbar, so sendet er ein ICMP-Echo-Response Paket zurück und signalisiert somit seine Erreichbarkeit. Es ist jedoch auch möglich, erwähnten Echo-Request mit falscher IP-Absendeadresse an die Broadcast-Adresse eines Netzsegmentes zu verschicken. Daraufhin senden alle ans Netzwerk angeschlossenen Rechner einen Echo-Response an die falsche IP-Adresse. Sollte der Angreifer nun eine große Anzahl von gefälschten IP-Paketen in einem kurzen Zeitraum (200 Pakete pro Sekunde) an diese Broadcast-Adresse versenden, so kann der Rechner hinter der gefälschten IP-Adresse unter dem ankommenden ICMP-Datenstrom zusammenbrechen und vom Netz getrennt werden. Unter Linux gibt es jedoch diverse Kernelparameter, die z.B diese Broadcasts ignorieren, sodass es nicht zu diesen Broadcast-Stürmen kommen kann.
Die Angriffe Smurf und Fraggle basieren auf den Broadcast-Angriffen. Smurf verwendet dazu die beschriebenen ICMP-Pakete und Fraggle UDP-Datagramme. Um einen erfolgreichen Angriff mit Smurf oder Fraggle zu starten, benötigt man Broadcast-Adressen, die auf die entsprechenden Daten-

pakete ebenfalls antworten. Solche Broadcast-Adressen werden Amplifier genannt. Tools, die einen Smurf Angriff durchführen, senden anschließend mittels IP-Spoofing gefälschte ICMP-Pakete an diesen Amplifier und generieren somit eine beträchtliche Anzahl von Antwortpaketen, die alle an die gefälschte Absendeadresse gerichtet sind und somit das Zielsystem überfluten. Dadurch wird am Zielsystem die komplette Netzwerkkapazität ausgelastet und der Rechner ist nicht mehr erreichbar. Mit dem bereits beschriebenen Portscanner Nmap kann man testen, ob ein Netzwerksegment als Amplifier missbraucht werden kann oder nicht.

Befehl zum Testen auf ICMP-Broadcast Anfragen:

```
Nmap -n -sP -PI -o is_amplifier.txt '141.55.*.0, 63, 64, 127, 128,
191, 192, 255'.
```

Darüber hinaus sollte man bei der bestehenden Gefahr von Smurf-Angriffen den Kernelparameter icmp_ignore_broadcasts aktivieren und bei Verdacht auf Fraggle Angriffe die Firewall so konfigurieren, dass UDP-Pakete verworfen werden. [14]

Befehl zum Ignorieren von Broadcast Angriffen:

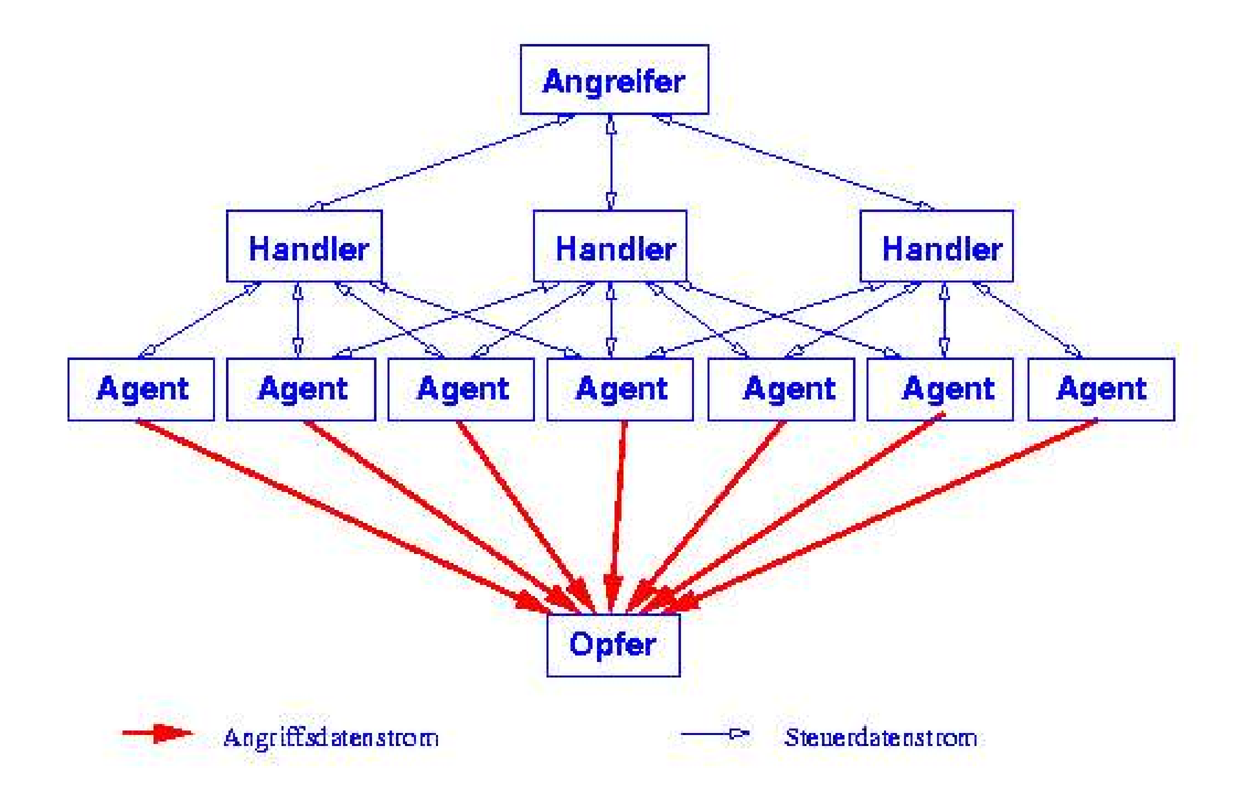
```
echo 1 > /proc/sys/net/ipv4/icmp_ignore_broadcasts.
```

Flooding Angriffe

Unter dem Begriff Flooding-Angriffe versteht man das tausendfache Versenden von Paketen an einen Zielhost mit der eigentlichen Absicht, die Netzwerkkapazitäten des angegriffenen Hostes auszulasten und den Host somit nicht mehr erreichbar zu machen. Beim IP-Flooding werden gefälschte IP-Pakte an den Zielhost versendet, beim SYN-Flooding sind dies TCP-Pakte mit gesetztem SYN-Flag. Im Gegensatz dazu kann man das User-Datagramm-Protokoll dazu missbrauchen, einen so genannten UDP-Flooding Angriff zu starten, indem man tausende von UDP-Paktete an einen Internetrechner versendet. [14]

Distributed Denial-of-Service -Angriffe

Im Gegensatz zu einfachen Denial-of-Service-Angriffen, bei denen oft nur von einem Rechner Datenpakte versendet werden, basieren Distributed-Denial of-Service-Angriffe auf einem Client/ Server-Modell. Hierbei werden die Clients auf verschiedene Rechner mit möglichst großer Bandbreite installiert und dann entsprechend ferngesteuert, um eine gemeinsame Angriffsbasis gegen ein Ziel zu schaffen. Im Bild Abbildung 5.1 ist sehr schön die Hierarchie eines solchen Angriffes zu sehen. Der Angreifer hat verschiedene DDoS-Clients auf verschiedene Rechner verteilt. Diese so genannten Handler senden die Angriffsaufforderung an die entsprechenden DDoS-Agenten weiter (z.B. trinoo oder Stacheldraht), die dann den entsprechenden Host mittels entsprechend besprochener Flooding Mechanismen angreifen. Auf eine ausführlichere Beschreibung bezüglich der Thematik Denial-of-Service wird hier verzichtet, weil es den Umfang der Bachelorarbeit zu sehr ausweiten würde.



aus [23]

Abbildung 3.3: DDoS-Angriffe mittels Agenten

3.6 Verteidigungsmaßnahmen gegen Denial-of-Service-Angriffe

Generell ist es sehr schwer, bei starken Denial-of-Service-Angriffen Abwehrmaßnahmen einzuleiten. Oft bleibt nur das Vorschalten von speziellen Filtern, die den schädlichen Traffic herausfiltern und somit die dahinter liegenden Rechner schützen.

anti_syn_flood.pl gegen Flooding Angriffe

Mit Hilfe des Programms anti_syn_flood.pl kann man spezielle Flooding-Angriffe abwehren, die so genannten TCP-SYN-Flooding-Angriffe. Das entwickelte Programm, welches im Anhang zu finden ist, überwacht den Datenverkehr auf auffällige TCP-Pakete. Wenn nun eine bestimmte Anzahl von SYN-Paketen das System erreicht oder verläßt, werden diese Verbindungen vom Programm automatisch zurückgesetzt, indem auf diese Anfrage jeweils ein TCP-Paket mit gesetztem FIN-Flag gesendet wird. Damit wird diese Verbindung abgebaut und verbraucht keinen Speicherplatz bei der Backlog Queue.

Scanner gegen Distributed Denial-of-Service-Agenten

Das Programm find_ddos ist ein Programm, mit dem man nach so genannten Distributed-Denial-of-Service-Agenten scannen kann. Es unterstützt die Suche nach bekannten Agenten für Stacheldraht, Trinoo und TFN. Gefundene Agenten werden gelöscht und somit ein Angriff verhindert. Weiterhin stellt eine ordentlich konfigurierte Firewall auch ein effektives Mittel gegen solche Angriffe dar.

4 Weitere Abwehrmaßnahmen in Netzwerken

4.1 Firewalls

Mit Hilfe von Internet Service Providern wie Arcor oder der Telekom verbinden sich heutzutage Millionen von Menschen mit dem Internet. An die ISP werden dabei hohe Anforderungen an die Verfügbarkeit, Stabilität und Sicherheit gestellt, die alle mit sehr hohen Kosten verbunden sind. Die bei den ISP eingesetzten Router unterstützen das Filtern des Netzverkehrs und würden somit bereits eine erste Schutzfunktion für den Enduser darstellen. Da die Filterung des ein- und ausgehenden Verkehrs jedoch mit einer erhöhten CPU-Last seitens der Internet Service Provider erkauft werden

würde, wird diese Sicherheitsoption meist nicht in Anspruch genommen. Es besteht jedoch die Möglichkeit, selbst solche Sicherheitsfunktionen zu nutzen, indem man eine Firewall aufsetzt und aktiviert. Im weiteren Verlauf soll somit kurz auf das Thema Firewalls, im Speziellen Firewalls auf Paketfilterbasis, eingangen und sich auf dieses Thema beschränkt werden, da im weiteren Verlauf eine Firewall auf Paketfilterbasis erstellt und in Betrieb genommen werden wird.[14]

4.1.1 Allgemeine Mechanismen

Firewalls kann man generell als Schnittstelle zwischen zwei Netzwerksegmenten ansehen, die passiert werden muss, um Rechner in einem anderen Netzwerk zu erreichen. Daraus lassen sich auch die Aufgaben einer Firewall ableiten. Sie soll erlaubte Netzaktivitäten durchlassen und nicht gestattete Kommunikationsbeziehungen verbieten und abblocken. Weiterhin ist es mit einer Firewall auch möglich, die Nutzeraktivitäten auf ein Netzsegment zu beschränken. Allgemein kann man folgende Anforderungsprofile an eine Firewall definieren: [14]

- jeglicher Datenverkehr geht über die Firewall
- nur laut Sicherheitsrichtlinien authorisierter Verkehr darf die Firewall passieren
- zur Realisierung der Firewall soll eine möglichst sichere Software eingesetzt werden
- jeder Datenverkehr, der nicht ausdrücklich erlaubt ist, soll geblockt werden
- Konfiguration und Wartung der Firewall soll nur über gesicherte Verbindungen laufen

Paketfilter im Allgemeinen werden auch Screening Router genannt und haben die Aufgabe, die Zugriffssteuerung auf Sockets nach IP-Adresse und Ports zu regeln. Dazu gibt es bestimmte Filterregeln, die im Paketfilter implmentiert sind. Unter Linux ist das Programmpaket IPTables die aktuelle Paketfiltersoftware. Diese Filterregeln werden von der eigenen Sicherheitspolitik abgeleitet und bestimmen z.B. ob generell UDP-Verkehr zugelassen oder verworfen wird und ob TCP-Dienste (mittels Ports) zugelassen werden oder nicht. Wichtig hierbei ist auch, dass der Paketfilter physischen Zugriff auf den I/O-Port des Netzwerkinterfaces hat und somit z.B. auch IP-Spoofing erkennen und abblocken kann. Die Konfiguration eines Paketfilters sollte in 3 Schritten erfolgen. Der erste Schritt umfaßt die Festlegung der eigenen Sicherheitspolitik und sollte bereits klären, welche Dienste erlaubt werden sollen und welche nicht aktiviert werden dürfen. Im zweiten Schritt werden die laut Sicherheitsrichtlinien zu filternden Pakettypen formal spezifiziert. Im folgenden dritten Abschnitt werden diese, in den formalen Spezifikationen angegebenen Anforderungen dann

in die Syntax der Paketfilter übertragen. [14]

4.1.2 Statische Paketfilter

Statische Paketfilter sind die einfachsten, zugleich funktionell am eingeschränktesten Firewall-lösungen. Sie prüfen die IP-Pakete anhand von:

- IP-Quell und Zieladressen
- TCP oder UDP Portnummern oder dem Funktionstyp (z.B. ICMP)
- ACK-Bit (neuer Verbindungsaufbau oder bestehende Kommunikationsbeziehung)

Anschließend wird entschieden, ob das Paket passieren darf oder nicht. Da hierbei nur ein einzelnes IP-Paket betrachtet wird und keine eigene Kontextfilterung stattfindet, kann die Firewall somit nur entscheiden, ob der zum IP-Paket gehörende Port geschlossen oder offen gelassen werden soll. Bei TCP-Verbindungen wird somit das erste Paket des Verbindungsaufbaus geprüft, bei dem das SYN-Flag gesetzt und das ACK-Flag nicht gesetzt ist, und dann entschieden, ob die Anfrage aus dem zu schützenden Portbereich kam oder nicht. UDP sollte man bis auf DNS, wenn benötigt, verbieten, da man sonst gegen UDP-Angriffe wie UDP-Flooding oder UDP-Spoofing nicht geschützt ist. Vorteil eines solchen Paketfilters ist der damit verbundene geringe finanzielle und zeitliche Aufwand zur Realisierung. Negativ ist jedoch, dass entscheidende Einschränkungen bei der Aufstellung von Filterregeln auftreten und keine Kontextfilterung durchgeführt werden kann. Dies können nur die dynamischen Paketfilter. [14]

4.1.3 Dynamische oder kontextbezogene Paketfilter

Dynamische Paketfilter können neben der Filterung auf IP- und TCP-Ebene auch kontextbezogene Daten verarbeiten, wobei auch die Filterung auf Anwendungsebene (z.B. FTP, MySQL, Java Server Pages) durchgeführt werden kann. Dies nennt man "stateful paketfiltering". Bei dem verbindungs-losen Protokoll UDP kann der statische Paketfilter nicht anhand von ACK-Bit oder des UDP-Headers erkennen, ob es sich hierbei um das erste Anfragepaket vom Client zum Server handelt. Dynamische Paketfilter hingegen speichern nach außen gesendete UDP- und TCP-Pakete in einer Liste ab und wissen aufgrund dieser Liste, welches UDP- bzw. TCP-Paket passieren darf. Legitime Antwortpakete müssen somit von demselben Rechner und Port kommen, an den die ursprüngliche Anfrage gesendet wurde. Darüber hinaus muss das Ziel derselbe Rechner und Port sein, von dem

die Anfrage kam. Dynamische Paketfilter speichern also TCP-Daten eines gesendeten Paketes ab und testen ankommende TCP-Pakete, ob die Zieladresse und der Zielport mit der Ursprungsadresse und dem Ursprungsport eines TCP-Paketes übereinstimmen, das zuvor gespeichert wurde. Ebenso müssen Ursprungsadresse und Ursprungsport eines ankommenden Paketes mit der Zieladresse und dem Zielport eines zuvor gesendeten Paketes übereinstimmen.
Da sich die Filterregeln dynamsich ändern können, spricht man von einem dynamischen Paketfilter. Darüber hinaus kann auch eine kontextbezogene Verarbeitung der Daten erfolgen und somit z.B. ein verdächtiger Code herausgefiltert werden. [14]

4.2 Virtuelle Private Netzwerke

Ein Virtuelles Privates Netzwerk stellt eine gesicherte oder ungesicherte Verbindung zwischen ein oder mehreren Netzwerken dar. Ein solches Virtuelles Privates Netzwerk, auch VPN genannt, setzt auf ein physisches Netzwerk auf. Ein VPN bietet Sicherheit über verschiedene kryptografische Protokolle an. Aus der Open Source Welt ist das Protokoll Ipsec bekannt oder auch die Programmsuite FreeS/WAN, die dieses Protokoll implementiert. VPN haben insgesamt 2 verschiedene Verwendungszwecke. Im ersten Fall verbinden sie getrennte Netzwerkbereiche desselben Netzwerkes. Wenn ein Unternehmen z.B. 2 verschiedene Zweigstellen hat, kann man das interne Netzwerk mit Hilfe eines VPN koppeln. Zweitens verbinden VPN mobile User mit dem Netzwerk der Firma, oft durch eine Firewall hindurch. So kann sich ein Benutzer z.B bei einem öffentlichen Internet Service Provider anmelden und sich anschließend mittels VPN-Software in das Firmennetzwerk einloggen und so remote arbeiten. [25]

4.3 Einbruchserkennungssysteme

Einbruchserkennungssysteme, auch Intrusion Detection Systeme genannt, überwachen ein Netzwerk nach einem bestimmten auffälligen Verhalten und schlagen Alarm, wenn ein Angriff startet oder bereits durchgeführt wird. Gute Intrusion Detection Systeme warnen nicht, wenn kein Angriff stattfindet, aber sie verpassen auch keine wirklichen Angriffe. Entsprechende Programme warnen den Administrator rechtzeitig über bestehende Probleme. Ein IDS kann auf 2 grundlegende Arten aufgebaut werden, einmal der Missbrauchserkennung und andererseits der Anomalieerkennnung. Bei der Missbrauchserkennung hält das IDS Ausschau nach bekannten und bestimmten Angriffsmustern und wertet eingehende Pakete dahingehend aus. Wenn ein Angriff auf Basis dieses Musters

erkannt wird, so schreitet das IDS ein und benachrichtigt den Administrator. Problematisch hierbei ist jedoch die Tatsache, dass, wenn ein Angriff auch nur minimal vom Bekannten abweicht, dies nicht von der Missbrauchserkennung bemerkt wird und der Angriff geht durch. Es ist bei diesem System also notwendig, immer auf dem neuesten Stand der Angriffstechnologien zu sein.
Der zweite Aspekt ist die Anomalieerkennnung. Hierbei überwacht das IDS das Netzwerk nach normalen Aktivitäten. Sollte jedoch eine unbekannte Aktivität durchgeführt werden, also eine Anomalie entdeckt werden, so meldet das IDS dies entsprechend und bietet im optimalen Fall auch gleich ein Gegenmittel an. Bekannte Open Source Programme sind Snort und Prelude. [25]

4.4 Honey Pots

Honey Pots, auch Netzwerkalarmanlagen genannt, sind kleine nützliche Programme, mit denen man ganze Netzwerke simulieren und aufstellen kann. Diese vorgetäuschten Netzwerke können dann eingerichtet werden und somit Angreifer anlocken, sodass man ihre Aktivitäten überwachen und mitprotokollieren kann. Dies ist nützlich für eine spätere Verfolgung des Angreifers. Honey Pots werden von Firmen bereits als vollständige Softwareapplikation herausgegeben, sodass man diese Netzwerke nur noch namentlich benennen müsste. Unter Linux ist die Software honeyd als Anwendung zum Aufsetzen von Honey Pots bekannt. [25]

4.5 Vulnerability-Scanner

Vulnerability-Scanner, kurz Scanner genannt, sind Programme, die ein System auf Schwachstellen überprüfen und entsprechende Berichte als Ausgabe produzieren.
Im Gegensatz zu wirklichen Einbrechern nutzen die Scanner die gefundenen Sicherheitslücken jedoch nicht komplett aus. Ansonsten würden diese Scanner die Netzwerkrechner lahm legen.
Vielmehr sollte man diese Scanner als Zusatzmerkmal in die eigenen Sicherheitsplanungen einbeziehen, da man mit ihnen sehr leicht nach verwundbaren Stellen im eigenen Netzwerk suchen kann, um die gefundenen Schwachstellen hinterher entsprechend zu schließen, indem man die dafür nötigen Sicherheispatches installiert. Ein sehr bekannter Open Source Scanner ist das bereits vorgestellte Programm Nessus, mit dem man nach bekannten Schwachstellen im Netzwerk suchen kann. [25]

4.6 E-Mail-Sicherheit

E-Mail wird heutzutage als vollwertiges Kommunikationsmittel akzeptiert. Jeder Benutzer kann sich ein E-Mail-Konto bei einem Freemail-Anbieter besorgen und anschließend beliebig viele E-Mails versenden. Da in das zugrundeliegende Protokoll Simple Mail Transfer Protocoll keinerlei Sicherheitsmaßnahmen eingebaut wurden, ist es einfach für Angreifer, die Absendeadresse der E-Mail zu fälschen und somit eine falsche Identität dem Empfänger der E-Mail vorzutäuschen. Darum gibt es heutzutage viele Produkte, die mit Hilfe kryptografischer Protokollle ein Mindestmaß an Sicherheit für die Kommunikation mittels E-Mail gewähren. Im Grunde läuft die Verschlüsselung nach folgender Prozedur ab:

- A erhält den öffentlichen Schlüssel von B
- A signiert ihre Nachricht mit ihrem privatem Schlüssel
- A verschlüsselt ihre Nachricht mit dem öffentlichen Schlüssel von B
- A sendet die verschlüsselte und signierte Nachricht an B
- B entschlüsselt die Nachricht mit seinem privaten Schlüssel
- B prüft die Signatur von A mit Hilfe des öffentlichen Schlüssels von A

4.7 Verschlüsselungen und sichere Kommunikationsprotokolle

Bei der Entscheidung, für oder gegen Verschlüsselung in der eigenen Firma, gibt es verschiedene Hinweise zu beachten. Auf der einen Seite sichert eine verschlüsselte Kommunikationsbeziehung den Datenverkehr gegen Mitlauscher und Angreifer ab, aber auf der anderen Seiten funktionieren dann Programme wie Contentfilter, Antivirenprogramme oder Firewalls nicht mehr richtig, wenn die komplette Kommunikation verschlüsselt wird. Hierbei muss man sich nun überlegen, ob eine Verschlüsselung notwendig ist oder nicht. Bei der Fernwartung von Rechnern hat sich das Programm Secure Shell als Quasistandard etabliert. Es ermöglicht, einen entfernten Rechner fern zu warten, wobei die komplette Datenverbindung mit starken Verschlüsselungsalgorithmen, wie z.B. AES verschlüsselt und somit gegen Mitlauschen oder Mitschneiden abgesichert wird.

5 Spezielle Sicherheitsmechanismen und Sicherheitspatches

Im nun folgenden Abschnitt sollen direkt die bekanntesten und aktuellsten Sicher-

heitstechniken vorgestellt werden, die es für das Betriebsystem Linux gibt. Darüber hinaus werden verschiedene Sicherheitsprodukte erläutert und diese tabellarisch mit den entsprechenden Vor- und Nachteilen gegenübergestellt. Im Konkreten sollen hier SE-Linux, Bastille-Linux, Openwall Security, Grsecurity und Exec-Shield beschrieben werden. Anschließende Hinweise geben Auskunft, für welchen Einsatzzweck welche Sicherheitssoftware am geeignetesten ist.

5.1 Softwarehärten bei Linuxrechnern

Aus dem Bereich Softwarehärtung unter Linux hat sich vor allem die Programmsuite Bastille-Linux hervorgehoben. Diese besteht aus einer Sammlung von Perl-Scripten, die nach dem Ausführen den User Stück für Stück darauf hinweisen, was als nächstes getan wird. Bastille-Linux sichert das System ab, indem es entsprechende Programme mit SETUID aufzeigt, das Netzwerk absichert, verschiedene Logging-Funktionen aktiviert und die schon beschriebenen Kernelhärtungen durchführt.
Folgende Einträge sollte in die Datei "/etc/sysctl.conf" vorgenommen werden, um die angesprochenen Schutzfunktionen im Kernel zu aktivieren:

```
net.ipv4.icmp_echo_ignore_all=1         # ignoriere ping anfragen
net.ipv4.tcp_syncookies=1               # verhindere syn-floods
net.ipv4.icmp_echo_ignore_broadcasts=1  # verhindere broadcasts
net.ipv4.conf.all.accept_source_route=0 # verhindere rip-spoofing
net.ipv4.conf.all.rp_filter=1           # verhindere spoofings
net.ipv4.ip_always_defrag=1             # Pakete immer defragment.
net.ipv4.conf.all.log_martins=1         # protokolliere spoofings
net.ipv4.conf.all.accept_redirects=0    # verwerfe icmp-redirects
```

.

Es sei darauf hingewiesen, dass das Programm Bastille Linux das Softwarehärten des gesamten Linuxsystems übernimmt.

5.2 Softwareseitige Emulierung des no-execution Flag

Aktuell gibt es zwei Möglichkeiten, um das bei einem Prozessor fehlende no-execution Flag zu emulieren: einerseits die Pageexec- und andererseits die Segmexec-Methode. Auf beide Methoden

soll im Weiteren detaillierter eingegangen werden.

Pageexec

Das Pageexec Verfahren nutzt die Aufteilung der Translation Lookaside Buffers (TLB) bei den Prozessoren ab der Pentium- und der K5-Reihe. Hierbei werden Daten und Code nicht in einem TLB verwaltet, sondern in zwei getrennten TLB`s für jeweils Code und Daten. Dabei sind die TLB als eine Art Bindeglied zwischen virtuellem und direktem physischen Speicher zu verstehen und wirken dabei als eine Art schneller Cache. Wenn der Prozessor auf eine virtuelle Speicheradresse zugreifen will, so schaut er zuerst im TLB nach, wird er hier nicht fündig, so wird die Adresse aus der Page Table in den Speicher geladen und zusätzlich noch in den TLB geschrieben. Normalerweise kümmern sich die x86-Prozessoren automatisch um das Caching der Einträge in den TLB, allerdings ist es auch der Software erlaubt, den kompletten TLB zu leeren oder einzelne Einträge für die entsprechenden virtuellen Adresseinträge zu löschen. Dieses Merkmal wird von der Pageexec-Technik ausgenutzt, um Non-Executable-Pages zu implementieren. Der Pageexec-Mechanismus löscht Speicherseiten, die nicht direkt ausführbar sein sollen aus dem TLB oder gewähren den Zugriff nur bei entsprechenden Supervisor-Rechten. Bei jedem Zugriff auf den TLB, der auf den geschützten Bereich abzielt, wird ein Page Fault vom Prozessor generiert und an den Kernel übergeben, der anschließend entscheiden muss, ob der Zugriff gültig oder nicht ist. [26]

Segmexec

Die zweite Art der Emulation des no-execution Flag ist das so genannte "segmentation based non-executable pages" (segmexec). Der Speicher wird hierbei in zwei Segmente geteilt, einmal das Daten- und andererseits das Code-Segment. Auf den entsprechenden x86-Prozessoren läuft Linux dann im Protected Mode und benutzt das Paging, welches bei jedem Datenzugriff, die aus Segment und Offset bestehende logische Adresse in eine virtuelle linerare Adresse umwandelt. Mit Hilfe dieser Technik wird der virtuelle auf dem physikalischen Speicher abgebildet. Normalerweise kann ein Prozess im Userspace 3 Gbyte an Speicher umfassen, wobei Segmexec diesen Bereich in zwei Hälften aufteilt. Die erste enthält alle Mappingmethoden für Schreib- und Lesezugriffe und die zweite Hälfte enthält alle Tabellen für den ausführbaren Code. Unter den ersten 1,5 Gbyte liegen also alle Daten, darüber die entsprechenden Einträge für den ausführbaren Code. Das Segment mit den ausführbaren Daten darf auf Mappings entsprechende Daten benutzen, umgekehrt darf dies

nicht geschehen. Mit Hilfe dieser Speichersegmentierung sind nun Code- und Datenbereiche unterscheidbar und das Laden von Instruktionen aus dem nicht ausführbar bestimmten Bereich führt zu einem Page Fault, den der Kernel entsprechend verarbeiten und somit den Zugriff unterbinden kann. Der Openwall Security Patch implemtiert den non-executable stack auf Basis des Segmexec-Verfahrens. [26]

Kombination von Pageexec und Segmexec

Der Exec-Shield Patch von Redhat und Pax, der Bestandteil des Security Patches Grsecurity ist, erweitert das bekannte Konzept des no-execution Flag. Neben der Funktionalität der non-executable stack area wird auch der Heap Speicher, der mit malloc() angefordert wird, geschützt. Folglich werden auch solche Datenbereiche gesichert, die ein Prozess mit Hilfe des Systemaufrufes mmap() in den Speicher abbilden. Grsecurity und somit auch Pax verwenden, je nach Hardware, sogar Pageexec oder Segmexec und können somit auf fast jeder Hardwarearchitektur laufen und das System entsprechend schützen. [26]

5.3 Linux Security Modules

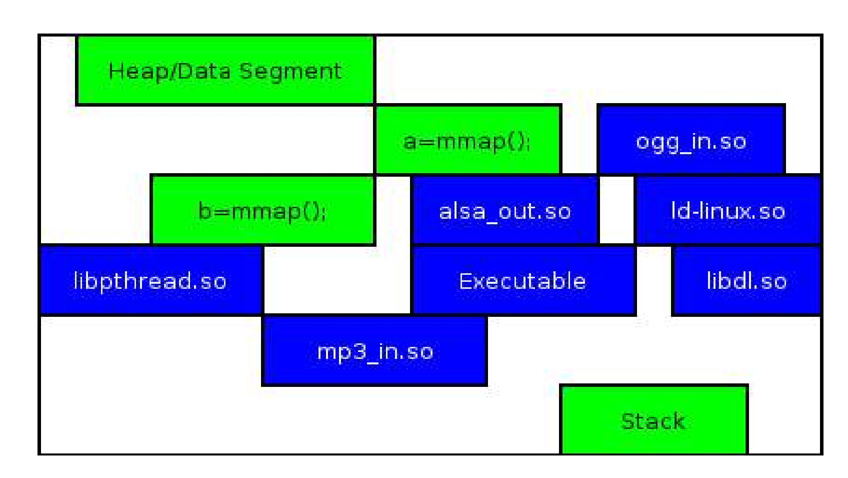
SE-Linux verwendet die Linux Security Modules-Architektur, um seine rigorose Policy-Politik mit Hilfe von Access Control Listen durchzusetzen. Einigen Mechanismen sind jedoch selbst die LSM nicht genug und sie greifen viel tiefer in das bestehende System ein. Dies geschieht bei Pax und Exec-Shield, denn sie sind eigenständige Patches und nicht nur Sicherheits-Module, wie die LSM. Die Kernelschnittstelle LSM ist insofern interessant, da sie sich unter anderem um die Zugriffsrechte der Speichersegmente eines Prozesses kümmert. Dabei handelt es sich um die wichtigen Funktionen mmap() und mprotext(), denn diese Funktionen sollen das ungewollte Einschleusen eines Codes in den Adressraum eines Tasks möglichst verhindern. Dabei gilt die Devise, dass ausführbare Bereiche nicht beschreibbar sein dürfen und umgekehrt. Dazu müssen folgende Aktionen unterbunden werden:

- "Erstellen von anonymen und ausführbaren Mappings
- Erstellen von Mappings auf gleichzeitig ausführbare und beschreibbare Bereiche eines Tasks
- Ändern eines ausführbaren Speicherbereiches als zusätzlich beschreibbar
- Ändern eines nicht ausführbaren, jedoch beschreibbaren Mappings als ausführbar"

[26]
Weiterhin gibt es eine Technik zur Verhinderung von Exploits, welche dafür sorgt, dass alle Mappings auf "PROT_EXEC"-Segmente in der so genannten ASCII-Amor-Area zu finden sind. Auf aktuellen 32-Bit Prozessoren sind das alle Bereiche von 0 bis 16 Mbyte. In diesem Bereich darf der vorhandene Code nicht hin und herspringen. Das würde geschehen, wenn per Buffer Overflow Code eingeschleust werden würde. Bei der Programmiersprache C kennzeichnet ein Null-Byte das Ende eines Strings und somit kann ein String selbst keine Null-Byte-Folge enthalten. Da der Angreifer seinen Exploit-Code jedoch als String übergeben muss und jede Adresse innerhalb der ASCII-Amor jedoch mindestens einen Null-Byte-String enhalten muss, kann der Exploit-Code dahin nicht springen. In solchen Fällen versuchen Exploits oft auf Adressen von Funktionen in Shared Libraries zuzugreifen, bei dem ASCII-Amor von Grescurity und Excec-Shield liegen jedoch diese Adressen ebenfalls im geschützen Bereich, sodass solche Exploits fehlschlagen. [26]

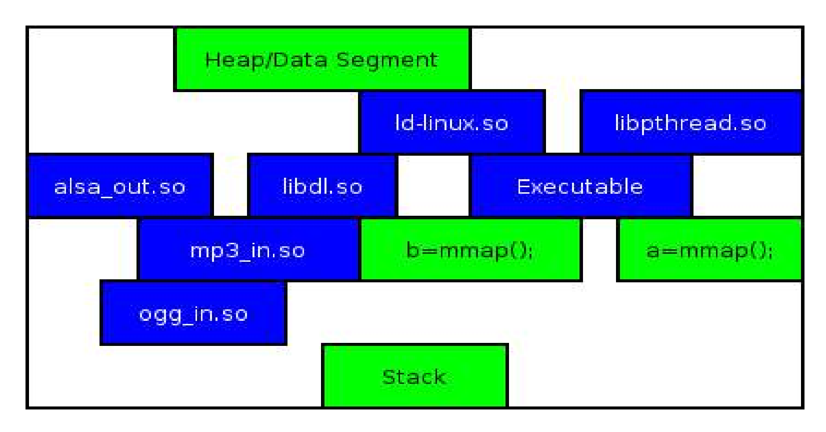
5.4 Address Space Layout Randomization

Address Space Layout Randomization ist eine Technik zum Verhindern von Exploits, bei der die Position der Speicherbreiche eines Tasks zufällig ausgewählt werden, sodass eine gezielte Manipulation von Daten durch den Angreifer fast unmöglich gemacht wird, da dieser nun erraten müsste, wo der Adress- und wo der Datenbreich im Programm anfängt und aufhört. [26]



aus [27]

Abbildung 5.1: Speicherbereich eines Programms im Task 1



aus [27]

Abbildung 5.2: Speicherbreiche desselben Programms im Task 2

5.5 Beschränkung der Zugriffsrechte

Mit dem Openwall und dem Grsecurity Patch wird eine weitere Angriffsquelle eliminiert, denn durch Aktivierung des Patches wird verhindert, dass ein Angreifer Daten und Prozesse im proc-Filesystem bearbeiten und auslesen kann, die ihm eigentlich nicht gehören. Nach dem Patch zeigen Programme wie "ps", "top" und "who" nur noch die eigenen Prozessdaten an, dadurch wird verhindert, dass ein Angreifer z.B. Daten mitlesen kann, die ihm eigentlich nicht gehören. Openwall bietet darüber hinaus noch die Möglichkeit, regulären Benutzern das Setzen von Hardlinks auf Dateien zu verbieten, für die sie keine Lese- und Schreibrechte besitzen. Dies könnte sonst von Exploits ausgenutzt werden, um Daten zu verändern und höhere Privilegien zu erlangen und verhindert damit auch die gefürchteten Race Conditions. Weiterhin bietet Openwall die Möglichkeit, den Schreibzugriff auf Named Pipes und Restricted Links zu begrenzen, sodass ein Benutzer ohne Schreibrechte auf diese Links oder in diese FIFOs entsprechend nicht schreiben darf. Grsecurity bietet darüber hinaus noch eine Beschränkung der Ausgabe des dmesg-Befehls an. [26]

5.6 Prozesslimits

Mit dem Systemaufruf setrlimit() und "RLIMIT_NPROC" kann man die Anzahl an verfügbaren Prozessen für einen Benutzer stark begrenzen. Jedoch wirkt dieser nur bei Prozessen, die mittles fork() erstellt wurden, wenn ein Benutzer seine reelle und aktuelle UID ändert, so wirkt dieses Mittel nicht und das System ist weiterhin anfällig für Fork Bomben. Mit Hilfe des Openwall Security Patches ist es jedoch möglich, das eben angesprochene Problem der Nichtbeachtung beim Content Wechsel eines Benutzers zu fokussieren und zu beheben. [26]

5.7 Access Control Lists

Unter Linux wird der Zugang zu Dateien und Verzeichnissen defaultmäßig über entsprechend gesetzte Datei-Zugriffsrechte gesetzt. Hierbei gibt es die drei Hauptrechte: Lese-(r), Schreib-(w), und Ausführungsrechte(x). Diese können mit dem Programm chmod geändert werden, wobei für neu erstellte Dateien und Verzeichnisse automatisch ein per umask bestimmter Oktalwert für die Rechte vergeben werden. Diese Art der Zugriffsrechteverwaltung bietet eine einfache Möglichkeit für die Verwaltung von Zugriffsrechten unter Linux, allerdings stößt diese Art auch schnell an ihre Grenzen.

Wenn man z.B. für einen Benutzer derselben Gruppe eine Datei freigeben möchte, so muss man das gleichzeitig auch für alle anderen Benutzer dieser Gruppe tun. Um dieses Problem zu umgehen, braucht man eine viel feingliedrigere Abstimmung der Zugriffsrechte, was mit Hilfe der so genannten Access Control Lists (ACL) implementiert wird. Mit Hilfe der ACLs ist es nun möglich, für jede Datei oder für jedes Verzeichnis eine Liste von sehr differenzierten Zugriffsrechten zu setzen.

Eine ACL eines Files oder Verzeichnisses kann somit folgende Einträge enthalten:

- "Zugriffsrechte für den namentlich aufgeführten Eigentümer
- Zugriffsrechte für die namentlich aufgeführte zugeordnete Gruppe
- Zugriffsrechte für alle anderen Benutzer des Systems
- Eine Maske für die Verteilung von Zugriffsrechten" [14]

Diese vier verschiedenen Einträge können nun mit Hilfe von speziellen ACL-Konfigurationsprogrammen bearbeitet und weitere Einträge vorgenommen werden, sodass diese mit entsprechenden Privilegien ausgestattet werden. Die zusätzlichen Einträge machen nun die eigentlichen ACL-Einträge aus. Diese Einträge geben an, ob ein Benutzer eine Datei lesen, schreiben oder ausführen darf. Die entsprechenden Rechte werden, wie schon bekannt, auch durch die Standard-Zugriffsrechte bestimmt, wobei die ACL die entsprechenden finalen, sprich maximalen, Rechte definiert. Wenn die Standardrechte einem Benutzer also Lese- und Schreibrechte zusichert, der Benutzer von der ACL jedoch nur Schreibrechte erhält, so kann der User die Datei entsprechend nur beschreiben und nicht lesen.

5.8 Weitere Sicherheitsmaßnahmen

Grsecurity bietet darüber hinaus noch viele weitere Features an, welche hier nur grob zusammengefasst werden sollen, wobei ich darauf verweise, hier nicht alle vollständigen Mechanismen zu erwähnen. Chroot Prozesse werden besonders geschützt, indem man verhindert, dass Prozesse innerhalb der Chroot Umgebung keine Setuid-Programme erstellen dürfen. Weiterhin ist es nicht erlaubt, in einer bestehenden Chroot Umgebung einen neuen Chroot Prozess zu erstellen. Darüber hinaus darf in dieser Umgebung kein Medium mit dem Befehl "mount" eingehängt werden.
Ein erweiterter Entropy Pool bei TCP-Verbindungen sorgt bei Grsecurity dafür, dass z.B. Sequenznummern von TCP-Verbindungen nicht so leicht erraten werden können und man damit gegenüber von Sequenznummer-Angriffen geschützt ist. [28]
Weiterhin muss erwähnt werden, dass die in Abschnitt 3 bereits beschriebenen Abwehrmaßnahmen, wie Antivirensoftware, Richtlinien für sichere Passwörter, Schutz vor Flooding Angriffen, Abblocken von Portscans hier ebenfalls eine wichtige Rolle spielen.

5.9 Sicherheitssoftware im Überblick

5.9.1 Openwall Security

Der Openwall Security Patch bietet folgende Optionsmöglichkeiten, den Linuxkernel zu härten:

- non-executable user stack area
- Automatische Erkennung und Emulation von GCC Trampolines
- Beschränkung der Links in /tmp
- Beschränkung der FIFOs in /tmp
- Beschränkung der Zugriffsrechte in /proc
- Durchführung des "RLIMIT_NPROC" Checks bei execve(2)-Systemaufruf
- Freigabe des nichtbenutzten Speichers bei der Verwendung von Shared Memory

Hierbei sind besonders die Punkte 1 und 3 interessant. Hinter dem "non-executable user stack area" verbirgt sich die Idee, den Stack für Userprogramme nicht ausführbar zu machen, damit normale Buffer Overflow Exploits scheitern. Da heutzutage die meisten Buffer Overflow Exploits darauf basieren, die Rücksprungadresse einer Funktion auf dem Stack dahingehend zu ändern, einge-

schleusten Code auszuführen, ist es möglich, solche Exploits an der Ausführung zu hindern, indem man den Stack "nicht-ausführbar" macht. Somit ist es schwieriger, diese Exploits auszunutzen. Darüber hinaus brauchen jedoch einige Programme einen ausführbaren Stack. Das wird mit dem Aktivieren der Option "Automatische Erkennung und Emulation von GCC Trampolines" gewährleistet. Weiterhin gibt es die Möglichkeit, für jedes Programm dieses Feature einzeln zu aktivieren mit dem beigelegten Programm chstk.c. Mit der Option "Beschränkung der Links in /tmp" kann man die Erstellung und Setzung von Links unter Linux dahingehend einschränken, dass, wenn nur ein User Lese- und Schreibrechte für eine Datei hat, er auch einen Hardlink darauf setzten darf. Damit verhindert man, dass z.B. ein Angreifer einen Link auf eine Datei setzt, die ihm nicht gehört, folglich auch kein Schaden durch versehentliches Schreiben auf diesen Link erfolgen kann. Diesem Beispiel folgt der Ansatz, der hinter der "Beschränkung der FIFOs in /tmp" steckt. Ein Benutzer darf nur dann in diese FIFO schreiben, wenn er auch die entsprechenden Lese- und Schreibzugriffsrechte für diese Datei besitzt. Mit dem Aktivieren des Features "Beschränkung der Zugriffsrechte in /proc" kann man verhindern, dass User Zugriffe auf fremde Prozesse in diesem Verzeichnis haben.
Wenn aktiviert wird, dann darf der User nur Dateien und Prozesse einsehen, zu denen er Lesezugriff hat. [28]

5.9.2 SE-Linux – Security Enhanced Linux

Security Enhanced Linux oder kurz SE-Linux genannt, wurde vom US-amerikanischen Nachrichtendienst NSA als Patch für den bestehenden Linux Kernel entwickelt. Nach der Einführung der LSM-Architektur wurde SE-Linux daraufhin angepasst und optimiert. SE-Linux ist aktueller Bestandteil des Linux Kernel der 2.6er Version.

Ziele

Das Linux Sicherheitskonzept basiert auf dem Konzept Discretionary Access Controll (DAC). Hierbei unterscheidet man zwischen zwei Benutzergruppen: diejenigen mit vollen Systemprivilegien und jenen mit eingeschränkten Rechten auf das System. Probleme gibt es, wenn ein Benutzer Dienste im System starten will, für die er Systemrechte benötigt, denn dann werden oft auf Ressourcen zugegriffen, auf die er sonst keinen Zugriff hätte. Wenn nun ein solches Programm oder ein solcher Dienst mit Systemprivilegien läuft und man in dem Programm Schwachstellen, wie z.B. Buffer Overflows, entdeckt, so kann ein Angreifer diese Lücke im System ausnutzen und auch meist

die entsprechenden Systemprivilegien erlangen. Hierbei macht sich das "Alles oder Nichts"-Prinzip der Zugriffskontrollmechanismen unter Linux negativ bemerkbar. Weitere Probleme des DAC sind, dass ein Benutzer völlig freie Hand hat, wie er die Rechte für seine Dateien setzt, denn das System kann hierbei nicht festlegen, welche Rechte er zu setzen hat. Hier greifen die Mandatory Access Control Mechanismen von SE-Linux, denn diese bieten im Gegensatz zum DAC dem System die Möglichkeit, selbst die Kontrolle über die Rechte zu übernehmen, ohne dass ein Benutzer diese umgehen kann. Das SE-Linux System greift hierbei auf eine Reihe von festgelegten Regeln, den Policy, zu. Weiterhin kennt SE-Linux den Zugriff auf Prozesse, hier Subjekt genannt, und Dateien, hier Objekt genannt. Mit Hilfe dieser Regeln ist es einem SE-Linux System sehr viel genauer möglich, einzelne Rechte für Programme, Dateien und Verzeichnisse zu setzen – nämlich genau die Rechte, die ein Prozess nur benötigt. Da all die beschriebenen Mechanismen im Kernelspace verankert sind, ist es nun auch nicht möglich, diese im Userspace zu umgehen. [30]

Architektur

Das SE-Linux basiert auf der Flask Security Architectur von 1999, für das es ein spezielles Betriebssystem gab. Die wichtigsten Kompontenten der Architektur sind der Security Server und der Object Manager. Von dem Security Server gibt es genau eine Instanz und hier werden zentral alle Zugriffsberechtigungsentscheidungen auf die entsprechenden Objekte getroffen, wobei man auf die vorhandene Policy zurückgreift. Der Security Server läuft als eigenständiger Prozess.
Der Objekt Manager ist für die Umsetzung der Zugriffsregelungen des Security Servers zuständig. Daraus ist es ersichtlich, dass eine Trennung zwischen Regelungen und Umsetzung der Zugriffsregelungen vorgenommen wurde. Der Object Manager wird durch die entsprechenden Kernel-Subsysteme wie Prozessverwaltung, Dateisystem usw. vertreten. Der Access Vector Cache (AVC) wird vom Object Manager verwendet, um Ergebnisse des Security Servers zwischenzuspeichern.
SE-Linux kennt insgesamt zwei Arten von Systemteilnehmern, einmal die Subjekte und anderseits die Objekte. Subjekte sind Prozesse, die im Linuxsystem laufen und auf entsprechende Objekte zugreifen können und wollen. Objekte hingegen sind Dateien oder Kernel-Objekte, die Netzwerkgeräte, Sockets und Files präsentieren. Die beiden eben genannten Dinge stellen die Grundlage für das Treffen von Entscheidungen des Security Servers im SE-Linux System dar, wobei die eigentliche Umsetzung der Entscheidungen vom Object Manager durchgeführt wird. Abbildung 5.3 stellt die Architektur von SE-Linux anschaulich dar.
Um die Objekte und Subjekte mit Security Labels zu markieren, werden die Security Fields der

LSM genutzt. Hierbei muss jedoch zwischen Security Context und Security Identifier (SID) unterschieden werden. Der Secuity Context ist ein String von unterschiedlicher, sprich variabler, Länge, der aus Benutzerkennung, Rolle und Domain besteht und durch einen Doppelpunkt getrennt ist. Diese Kennung wird dem User beim Login zugewiesen. Der Security Identifier ist ein 32-Bit integer Wert.

Objekte bekommen bei ihrer Erzeugung eine solche SID, die vom Security Manager generiert wird. Für Dateien werden die SID nicht verwendet, für diese gibt es persistente SID, die einen Neustart des Systems überstehen. Im Security Server findet eine Zuweisung zwischen SID und Security Context statt und wenn nun ein Subjekt auf ein Objekt zugreifen will, so wird über das AVC bei dem Security Server nachgefragt, ob der Zugriff rechtens ist. An den Server werden hierbei die SID des betreffenden Objektes und Subjektes sowie die Objektklasse übergeben und dieser berechnet nun, ob der Zugriff gestattet ist. Als Antwort gibt der Security Server einen so genannten Zugriffsvektor, bestehend aus den beiden SID, der Klasse sowie den berechneten Zugriffsrechten zurück.

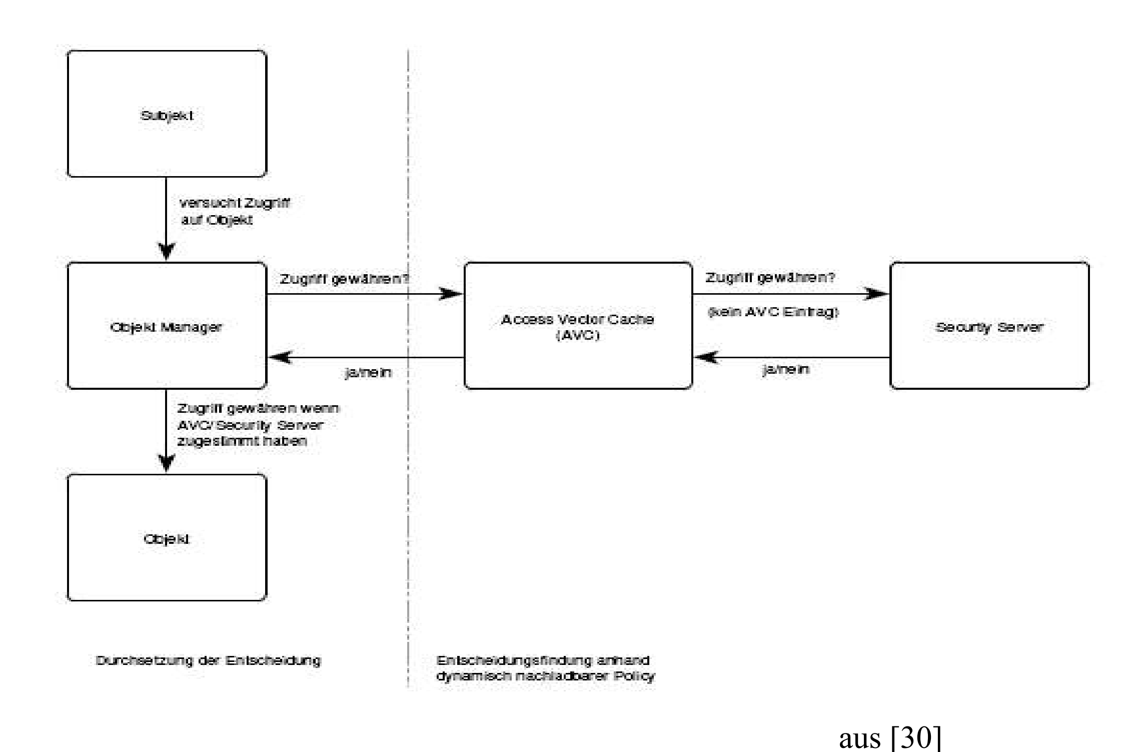
Natürlich gibt es auch die Möglichkeit, die Policy-Regeln zu verändern. Hierbei werden dann die aktuellen AVC-Einträge für ungültig erklärt und anschließend die Einträge in der AVC nach den neuen Policy Regeln aktualisiert. Dann werden mittels Callback- Funktionen die Object Manager über die Neuerungen informiert und diese wiederum aktualisieren die Rechte für die Kernel-Objekte und anschließend informiert der AVC den Security Server, dass die neue Policy aktiv ist. Es wurde erwähnt, dass die Benutzerkennung beim Login zugewiesen wurde und nichts mit der eigentlichen Linux UID zu tun hat, denn auch, wenn der Benutzer seine UID , z.B. mittels su ändert, bleibt die SE-Linux Nutzerkennungsmarke dieselbe. Die Implementierung von SE-Linux stützt sich auf die bereits erwähnten Linux Security Modules.

Anwendungen

Für eine Policy werden bei SE-Linux verschiedene Modelle eingesetzt: einmal die bereits bekannte Benutzerkennung, zweitens die Type Enforcement (TE) und drittens das Role-Based Access Control (RBAC).

Das Type Enforcement sorgt dafür, das Subjekte entsprechender Domains und Objekte zu Typen zugeordnet werden. Die Zusammenhänge zwischen Domain und Typen werden im Security Server als Matrix dargestellt und können in beliebiger Anzahl erstellt werden, sodass sehr feine Abstimmungen zwischen Subjekten und Objekten entstehen können. Intern im Security Server wird jedoch kein Unterscheid zwischen Domain und Typ gemacht, sie dienen nur der sprachlichen

Unterscheidung für den Menschen.
Das Role-Based Access Control hingegen weist jedem Benutzer eine oder mehrere Rollen zu, wobei jede Rolle wiederum verschiedenen Domainen zugordnet werden. Jede Rolle hat die für ihre zugeteilten Domainen bestimmten Zugriffsrechte, die auch mit newrole durch den autorisierten Benutzer geändert werden können. Die kann natürlich nur in dem von ihm zulässigen Rahmen erfolgen. Vorhandene Domain und Typen werden in entsprechenden Konfigurationsdateien beschrieben, wobei hier natürlich explizit beschrieben wird, welche Domain wie und auf welchen Typ zugreifen kann und darf. Hierbei gilt das Prinzip, dass alles, was nicht ausdrücklich erlaubt ist, verboten ist.
Weiterhin gibt es Konfigurationsdateien, in denen der Zugriff zwischen Typen und Daten beschrieben wird. Dies wird File Context genannt. Normalerweise kann, im Gegensatz zur Rolle, die Domain nicht geändert werden, dies geschieht automatisch und wird als Type Transitions bezeichnet, die vorher explizit in den Policy festgeschrieben werden muss. Solche Wechsel passieren z.B. wenn ein Dienst mittels init gestartet wurde, aber eigentlich nicht die vollen Systemprivilegien braucht. Durch ein Domainwechsel wird hierbei erreicht, dass der Dienst nicht mit den gefährlichen Systemprivilegien, sondern nur mit den benötigten Privilegien abläuft.



aus [30]

Abbildung 5.3 Architektur von SE-Linux

Beispiele für SE-Linux Regelwerk

Regeln für die Typen und Domainen werden in Konfigurationsdateien beschrieben:

```
allow type_8 type_9:class { perm_1 ... perm_n };
```

Objekte können darüber hinaus auch zu Default-Typen werden:

```
type_transition type_8 type_9:class
default_type
```

Erlaubte Typen pro Rolle werden wie folgt konfiguriert:

```
role rolename types { type_8 ... type_n };
```

Erlaubte Rollen pro Benutzer:

```
user username roles { role_1 .. role_n };
```

Rollenwechsel werden so konfiguriert:

```
role_transition current_role program_type new_role;
```

Beispielregeln für den Start von Prozessen:

```
allow user_t user_openoffice_t:process { sigkill sigstop signal };
allow user_openoffice_t user_t: { sigchld };
```

Einige Regeln für die Benutzerverwaltung:

```
user root roles { sysadm_r user_r };
user knoppix roles { user_r };
```

Der Standardtyp für Rollen sieht so aus:

```
user_r: user_t
sysadm_r: sysadm_t
system_r: system_t
```

Wollte man dem BenutzerKnoppix systemweite Privilegien geben, so würde man seinen Regelsatz dahingehend ändern, dass man sagen würde:

```
user knoppix roles { sysadm_r user_r };
```

Beispiel für das Starten von Openoffice:

```
bash> openoffice &
```

Daraus ergeben sich folgende Sicherheitskontexte:

Für die Shell:

`user_u:user_t:shell_t` und für Openoffice:

`user_u:user_r:openoffice_exec_t`

Daraus folgt nun der Regelsatz zum Starten von Openoffice mittels Bash:

```
allow shell_t openoffice_exec_t:process execute;
```

5.9.3 Grsecurity

Die Grsecurity Patch-Suite wird von Brad Spengler und Michael Dalton gewartet und weiterentwickelt und ist unter [28] zu beziehen. Neben den bereits beschriebenen wichtigen Features möchte ich hier noch einmal auf die Wichtigsten eingehen, da sich diese teilweise drastisch von den anderen Security Patches abheben und somit doch einer gesonderten Erwähnung verdienen. Im Einzelnen möchte ich auf folgende Merkmale eingehen:

- Access Control Lists für IP
- Erweiterte Address Space Randomization Features
- Dateisystem Schutzfunktionen
- Netzwerkbasierende Schutzfunktionen

Mit Hilfe der IP Access Control Lists kann ein Administrator folgende Dinge festlegen:

- welche IP und Port ein Prozess benutzen darf, um sich am Server zu binden
- auf welche IP und Port sich ein Anwender remote verbinden darf
- welche Art von Sockets ein Prozess nutzen darf
- welche Protokolle Sockets nutzen dürfen

Grsecurity unterstützt folgende Address Space Randomization Features:

- Randomized Kernel Stack Base
- Randomized mmap() Base
- Randomized User Stack Base

Bei dem ersten beschriebenen Mittel werden bei jedem System Call die Kernel Stack Adressen zufällig ausgewürfelt. Beim zweiten werden die Positionen der Dynamischen Shared Libraries neu verteilt und beim dritten wird die Adresse des Userland Stack bei jedem Aufruf neu erstellt. Grsecurity unterstützt daneben die so genannte "emulate trampolines"-Technik, mit der es möglich ist, für Anwendungen kurzzeitig ausführbar auf den Speicher im nicht-ausführbaren Bereich zuzugreifen.

Mit dem geschützten System Call mprotect() wird verhindert, dass eine Anwendung zwischen verschiedenen Pageing Schutzmaßnahmen hin und her wechselt. Darüber hinaus wird der Schreibzugriff auf "/dev/mem" und "/dev/port" verboten, sodass es für einen Angreifer nur noch schwer möglich ist, schädlichen Code in den laufenden Kernel einzubringen. Weiterhin werden Kernel Symbole versteckt, sodass kontrolliert werden kann, wer Informationen über laufende Kernel Symbole und geladene Module abrufen darf. Grsecurity verfügt über folgende Dateisystem Schutzfunktionen:

- Linking Beschränkungen
- FIFO Beschränkungen
- Chroot Beschränkungen

Die Linking Beschränkungen bewirken, dass ein Benutzer keine harten Links auf Dateien setzen kann, die ihm nicht gehören. Anwender dürfen nur noch symbolischen Links in ihnen zugänglichen Verzeichnissen folgen oder wenn sie der Besitzer der Datei oder des Verzeichnisses sind. Die Chroot Beschränkungen von Grsecurity erschweren es einem Angreifer, aus einer Chroot Umgebung auszubrechen und somit Schaden im gesamten System zu verhindern.

Der Standard TCP-Stack unter Linux ist sehr anfällig für Angriffe, die z.B. im Erraten von Sequenznummern liegen. Grsecurity nutzt einen größeren Zufallspol, Entrophy Pool genannt, um die initierende Sequenznummer möglichst nicht durch Erraten zu erlangen. Weiterhin werden zufällige IP Identifications eingesetzt, um somit das so genannte Betriebssystem Fingerprinting zu umgehen. Weiterhin besteht die Möglichkeit, den Zugriff auf Sockets zu kontrollieren. Es ist möglich, einer Gruppe den Zugriff auf Sockets zu untersagen sowie bestimmten Server und Clients einer Gruppe den Zugriff auf Sockets zu unterbinden.

5.9.4 Sicherheitskonzepte für verschiedene Risikogruppen

Ich möchte nun 3 Szenarien aufstellen für 3 verschiedene Anforderungen an die Sicherheit eines Linuxsystems:

- Einzeluser Linux Betriebsystem
- Entwicklungsumgebung im Multiuser Betrieb
- Internet Service Provider (ISP)

Für das Einzeluser-System empfiehlt sich mindestens der Einsatz einer Firewall auf Basis von IP-Tables, auf die im weiteren Verlauf der Arbeit noch eingangen werden soll.
Für die Entwicklungsumgebung im Multiuser Betrieb wird der Einsatz von Libsafe zur Vermeidung von Buffer Overflows empfohlen. Darüber hinaus sollte noch der Openwall Security Patch eingesetzt werden, um das System gegen Exploitversuche zu schützen. Darüber hinaus erlaubt SE-Linux eine sehr fein gestaffelte Anpassung der Zugriffsrechte und sollte somit in den Kernel eingebaut werden. Sollten die Systeme ans Internet angeschlossen sein, empfielt sich der Einsatz von Bastille Linux zur Gesamthärtung des Systems sowie der Einsatz einer Firewall.
Für den Internet Service Provider empfiehlt sich, die Grsecurity Patchsuite einzusetzen, zusätzlich noch das System mit Bastille Linux abzuhärten und je nach Bedarf eine Firewall einzusetzen. Da Grsecurity die meisten sinnvollen Features in sich vereint, sollte man ein Linuxsystem auf Basis von Grsecurity bevorzugen gegenüber einem System, das mit Openwall gehärtet wurde.
Die Features sprechen hier eindeutig die Sprache für den Einsatz von Grsecurity. Sollte es jedoch nötig sein, eine sehr feine Abstimmung der Zugriffsrechte zu setzen, so muss zusätzlich SE-Linux eingesetzt werden.
Im Großen und Ganzen belibt zu erwähnen, dass man mit kombinierter Hilfe von Bastille Linux, Grsecurity, SE-Linux und einer Firewall ein sehr sicheres Linuxsystem aufbauen kann, zusätzlich sollte man die unter 3. beschriebenen Maßnahmen durchführen, um schließlich ein Linuxsystem aufzusetzen, welches gegen viele Angriffe gewappnet ist.

5.9.5 Sicherheitsmechanismen in tabellarischem Überblick

Sicherheitsmerkmale für Linuxsysteme	SE-Linux	Openwall	Bastille Linux	Grsecurity	Exec-Shield	Firewall
Softwarehärtung			X			
Pageexec				X	X	
Segmexec		X		X	X	
Kombination von Pageexec und Segmexec				X	X	
Schutz vor Heap Overflows Exploits				X		
Schutz vor Buffer Overflow Explits		X		X	X	
Linux Security Modules Funktionalität	X					
Address Space Layout Randomization				X		
Erweiterte Address Space Randomization Features				X		
Beschränkte Zugriffe auf Links		X		X		
Beschränkte Zugriffe auf FIFOs		X		X		
Beschränkte Zugriffe innerhalb Chroot				X		
Beschränkte Zugriffe auf Dmesg				X		
Beschränkung der Zugriffsrechte für /proc		X		X		
Steuerung der Funktionalitäten über sysctl-Aufrufe				X		
Prozesslimits		X		X		
Erweiterte Logging-Funktionen				X		
Vorbeugung gegen Bruteforcing Maßnahmen				X		
Shared Memory Protection		X		X		
Erweiterterung des Entropy Pool				X		
Zugriffsteuerung mittels Access Control Lists	X			X		
Access Control Lists für IP				X		
Schutz vor TCP-,IP-,und UDP-Angriffen						X

Abbildung 5.4: Sicherheitsmechanismen und Konzepte im Überblick

6 Implementation eines sicheren Linuxrechners

Die Implementation des sicheren Linuxrechners soll sich hierbei auf ein vorhandenes Linuxsystem stüzten, das auf Basis von Debian arbeitet. Als Grundlage soll hierbei die Distribution Knoppix genutz werden, weil es weit verbreitet ist und man es sehr leicht erweitern und modifizieren kann.
Die Implementation stützt sich hierbei auf verschiedene Eckpfeiler der Sicherheit. Der grundlegendste Punkt ist die Absicherung des Kernels durch den Sicherheitspatch Grsecurity und durch Aktivieren von Kernelparametern. Weiterhin werden Serverdienste mit Hilfe des TCP-Wrappers geschützt und Portscans mit Hilfe von Patches und Portsentry aktiv verhindert. Die Firewall auf Basis von IPTables verhindert Angriffe auf IP, UDP und TCP-Ebene und das Programm anti_syn_flood.pl schützt den Server vor SYN-Flooding Angriffen. Der Webserver wird mit Hilfe von ModSecurity gegen webbasierende Angriffe gesichert, zusätzlich wird Software gegen Schadprogramme wie Viren, Würmer, Trojaner und Rootkits eingesetzt und das System mit Bastille Linux zusätzlich gesichert. Um die Serverumgebung gegenüber von Buffer Overflows

abzusichern wird zusätzlich Libsafe eingesetzt und mit Hilfe des Programms Nessus werden Systemschwachstellen aufgedeckt.
Schließlich wird das System mit dem Intrusion Detection System Snort ausgestattet und mit Syslogd alle Vorgänge auf dem Linuxserver überwacht und protokolliert.
Das sichere Linuxsystem soll Grsecurity nutzen. Der Verzicht auf Openwall und den Exec Shield ist damit zu begründen, weil Grsecurity die meisten nützlichen Features vereint. Einzelne Features können im laufenden Betrieb mittels sysctl-Aufrufen direkt im Kernel geändert werden. Für Grsecurity sprechen auch die unterschiedlichen Logging-Funktionen, mit denen der Administrator das eigene System gut überwachen kann. Da Grsecurity schon detailliert beschrieben wurde, soll an dieser Stelle auf Abschnitt 5 verwiesen werden..

6.1 Aktivierung von Kernel Parametern

Im Folgenden werden nun Möglichkeiten zur Verbesserung der Systemsicherheit aufgezeigt, indem bestimmte Systemparameter im Verzeichnis "/proc/sys/net/ipv4" gesetzt bzw. nicht gesetzt werden. [14]

Der icmp_echo_ignore_all-Parameter

Der Parameter icmp_echo_ignore_all dient dazu, festzulegen, ob der Linuxkernel auf ICMP-Anfragen von fremden Rechnern antworten soll oder nicht. Ist dieser Parameter gesetzt, so ignoriert der Kernel alle ICMP-Anfragen und sendet entsprechend auch keine ICMP-Pakete zurück. Dieser Parameter ist also sinnvoll, wenn man seinen Rechner gegen ICMP-Angriffe, wie z.B. ICMP-Flooding schützen will.
Befehl zum Aktivieren des Parameters:

```
echo 1 > /proc/sys/net/ipv4/icmp_echo_ignore_all
```

Der tcp_syncookies-Parameter

Bei einem SYN-Flooding Angriff kommen tausende von Verbindungsanfragen auf den eigenen Rechner zu und verstopfen somit die interne Backlog-Queue, die eingehende Verbindungsanfragen protokolliert und speichert.
Zur Vorbeugung eines SYN-Flooding Angriffes sollte man somit den internen Backlog nach oben

setzen. Zusätzlich sollte man dann entsprechend den tcp_syncookies-Parameter setzen. [14]

Befehl zum Aktivieren der Parameter:

```
echo 2048 > /proc/sys/net.ipv4.tcp_max_syn_backlog
echo 1 > /proc/sys/net/ipv4/tcp_syncookies
```

Der icmp_ignore_broadcasts-Parameter

Ein weiteres Problem stellen so genannte ICMP-Broadcast-Stürme dar, die als Denial-of-Service-Angriff zur Folge haben können, dass der betreffende Host vom Netzwerk getrennt wird. ICMP-Broadcasts entstehen, wenn Angreifer ein manipuliertes Datenpaket an die Broadcast Adresse des entsprechenden Subnetzes schicken. Wenn sich nun z.B. 230 Rechner in diesem Subnetz befinden, antworten all diese Rechner auf das manipulierte Paket und überschwemmen somit den betreffenden Host. Dies gipfelt in einem ICMP-Flooding, wenn der Angreifer ca. 100 Pakete pro Sekunde an die Broadcast-Adresse versendet. Mit dem Kernelparameter icmp_ignore_broadcasts kann man sich gegen solche Broadcast-Angriffe schützen. [14]

Befehl zum Aktivieren des Parameters:

```
echo 1 > /proc/sys/net/ipv4/icmp_ignore_broadcasts
```

Der accept_source_route-Parameter

Das Routing Information Protokoll (RIP) wird zur Verbreitung der Routingtabellen von Rechner und Router benötigt. Hierbei versendet der Router diese Tabellen mittels Broadcast an benachbarte Rechner, jedoch findet keinerlei Authentifizierung statt, sodass es mittels RIP-Spoofing möglich ist, ganze Kommunikationsbeziehungen zwischen Rechner zu manipulieren. Mit dem accept_-source_route-Parameter ist es nun möglich, solchen RIP-Spoofing, auch RIP-Redirection Angriff genannt, vorzubeugen.

Befehl zum Aktivieren des Parameters:

```
for i in /proc/sys/net/ipv4/conf/*/accept_source_route; do echo 0
> $i done
```

Dieser oben genannte Befehl deaktiviert das Source-Routing Feature und unterbindet solche Angriffe schließlich. [14]

Der rp_filter-Parameter

Ein weiterer gefährlicher Angriff ist das so genannte IP-Spoofing, dass meistens im Zusammenhang mit Denial-of-Service angewendet wird. Gelingt es einem Angreifer, gespoofte Pakete in das eigene Netzwerk einzuschleusen, so kann das als ein Zeichen für einen bald folgenden Angriff gesehen werden.

Befehl zum Aktivieren des Parameters:

```
echo 2 > /proc/sys/net/ipv4/conf/eth0/rp_filter
/etc/init.d/networking restart
```

Danach muss man die Netzwerkimplementation des Linuxrechers neu starten. Dies passiert mit dem 2. Befehl. [14]

Der ip_always_defrag-Parameter

Es ist möglich, jedes IP-Paket zu fragmentieren und danach entsprechend zu versenden. Bei der Fragmention wird im normalen Fall das IP-Paket so geteilt, dass der originale Port, der Zielport und die TCP-Flags im ersten fragmentierten Paket versendet werden. Wenn man jedoch die Fragmention so einstellt, dass nur die Portangaben, nicht jedoch die TCP-Flag Informationen, in das erste Paket passen, so kann man damit das Aufdecken eines Scanvorganges verhindern. Der Angreifer verschickt somit den TCP-Header über mehrere Pakete, um Intrusion-Detection-Systeme und andere Anti-Scanning Programme zu hintergehen. Man kann diese Art der Verschleierung jedoch aufdecken, wenn jedes IP-Datagramm aufgeschlüsselt und defragmentiert wird. [14]

Befehl zum Aktivieren des Parameters:

```
echo 1 > /proc/sys/net/ipv4/ip_always_defrag
```

Der genannte Befehl schaltet diese Schutzfunktion im Linuxkernel ein. [14]

Der log_martins-Parameter

Mittels des log_martins-Parameter kann man alle gespooften Source-Routed und Redirect-Pakete mitloggen. [14]

Befehl zum Aktivieren des Parameters:

```
for i in /proc/sys/net/ipv4/conf/*/log_martins; do  echo 1 > $i
done
```

Der accept_redirects-Parameter

Eigentlich werden ICMP-Redirekt Pakete von Routern dazu verwendet, einem Internetrechner eine bestimmte Route vorzugeben, auf der dann die entsprechenden Kommunikationspakete ausgetauscht werden. Wenn es nun einem Angreifer gelingen sollte, solche ICMP-Redirect Pakete zu fälschen, so kann dies für einen anschließenden Man-in-the-Middle Angriff genutzt werden. Man deaktiviert das Empfangen von Redirect-Paketen mit dem unten genannten Linux Befehl. [14]

Befehl zum Aktivieren des Parameters:

```
for i in /proc/sys/net/ipv4/conf/*/accecpt_redirects; do echo 0 >
$i done
```

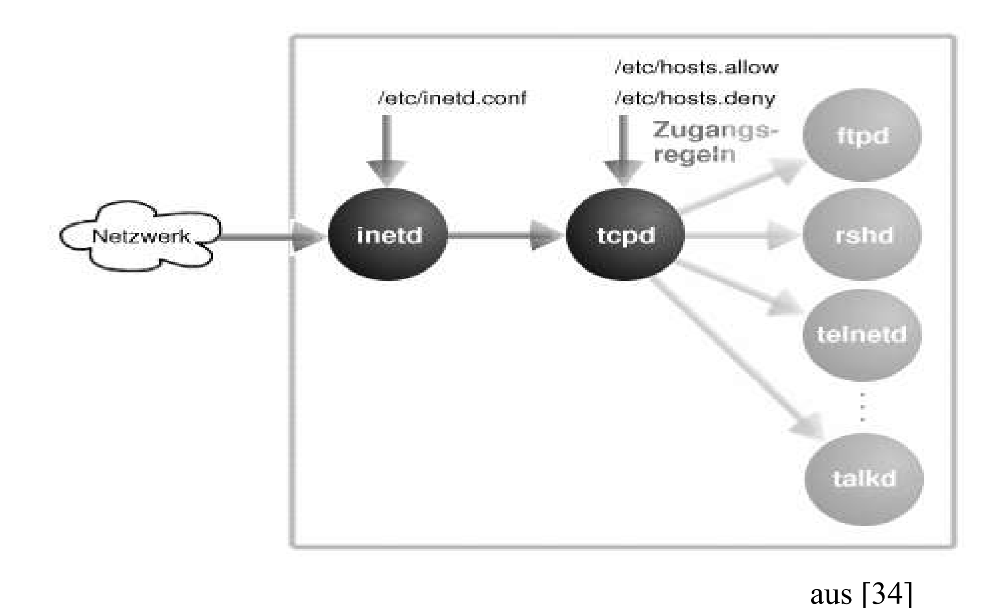
6.2 Schutz von Systemdiensten mittels TCP-Wrapper

Verbindungen zum Linuxserver erfolgen normalerweise auf direktem Weg, ohne Protokoll oder Sicherungsschicht zwischen dem sich verbindenden Client und dem Server. Serverdienste werden

bei einem modernen Linuxbetriebssystem mittels xinetd gestartet und aktiviert.
Einträge zum Starten des Dienstes ssh würden wie folgt aussehen:

```
service timed
{
        flags           = REUSE
        socket_type     = stream
        wait            = no
        user            = root
        server          = /usr/sbin/timed
        log_on_failure  += USERID
        disable         = no
}
```

Verbindungsanfragen werden hierbei direkt an die Serversoftware auf dem Linuxrechner weitergeleitet.
Der TCP-Wrapper kann von [37] bezogen werden. Die Verwendung eines TCP-Wrappers wird empfohlen, da hiermit 2 Vorteile gegenüber dem herkömmlichen xinetd einhergehen. Als erstes legt sich der TCP-Wrapper wie eine Art Schild um den zu schützenden Dienst. Mittels Syslog kann darüber hinaus protokolliert werden, wer sich zu diesem Dienst verbunden hat. Weiterhin arbeitet der TCP-Wrapper autonom im Hintergrund, sodass Client-Anwendungen den Wrapper nicht bemerken und somit keine Beeinträchtigung des Dienstes stattfindet. Der eigentliche Sinn eines TCP-Wrappers besteht jedoch in seinem Zugriffs-schutz.
Hierbei wird unter Linux auf die Dateien /etc/hosts.allow und /etc/hosts.deny zugegriffen, die die Zugriffsbestimmungen für das System definieren. Die Datei /etc/hosts.allow legt fest, welcher Dienst für welchen Host erreichbar ist. Im Gegensatz dazu werden in /etc/hosts.deny alle Dienst/Host-Kombinationen aufgeführt, die einen Zugriff auf Systemdienste haben. Zum Beispiel kann Port-sentry für einen Angreifer den Zugriff auf die Serverdienste verbieten, indem es die Datei /etc/hosts.deny entsprechend modifiziert und der TCP-Wrapper diese Einsellungen für die Dienste dann übernimmt.
Die Datei /etc/hosts.deny sollte den Eintrag "ALL:ALL" enthalten. Dieser legt fest, dass erst einmal alle Verbindungen verboten werden. Danach kann man Schritt für Schritt die Zugriffsbestimmungen in der /etc/hosts.allow festelegen. Im folgenden Beispiel wird die Datei /etc /hosts.allow dahingehend modifiziert, dass sie jedem Benutzer von der Domain sicheresnetzwerk.org erlaubt, den Dienst "E-Mail" auf dem Server zu nutzen: "sendmail: @sicheresnetzwerk.org".
Um den TCP-Wrapper über xinetd für das oben genannte Beispiel zu aktivieren, genügt es, den Eintrag "server = /usr/sbin/timed" durch den Eintrag "server = /usr/sbin/tcpd /usr/sbin/timed" zu ersetzen. Die folgende Abbildung verdeutlicht die Funktionsweise des TCP-Wrappers. [34]



aus [34]

Abbildung 6.1: TCP-Wrapper kontrolliert Zugriffe auf Serverdienste

6.3 Mögliche Verhinderung von Portscans

Neben der schon beschriebenen Methode des Kernelpatchens gegen Portscans auf Basis von FIN-, XMAS- und NULL-Flags bei TCP-Paketen, gibt es noch die Möglichkeit, den Kernelpatch Grsecurity einzuspielen. Danach muss noch die Paketfiltersoftware IPTable mit dem von Grsecurity mitgelieferten Patch modifiziert werden. Es ist bekannt, dass solche gesicherten Systeme bei einem Portscanversuch dem scannenden Host in bestimmten Zeitintervallen immer mal wieder kleine Pakete als Antwort darauf generieren und somit der angreifende Host gezwungen wird, kontinuierlich diese Verbindungen in der Backlog Queue offen zu halten. So kommt es anschließend beim Erreichen des Limits in der Backlog Queue des Kernels dazu, dass keine neuen Verbindungen mehr aufgebaut werden können - ein gutes Beispiel für einen passiven Denial-of-Service Angriff vom angegriffenen Host gegen den Angreifer selbst. Allerdings soll dies hier nur am Rande erwähnt werden. Durch den eben beschriebenen Vorgang kommt es dazu, dass es theoretisch unendlich lange dauert, bis der Portscanner seine Ergebnisse vorlegen kann. Ein Test hat gezeigt, dass diese geschützten Systeme auch nach mehreren Tagen des Scannens keine Ergebnisse erzielen – Portscans verpuffen also in Wirkungslosigkeit. Dies wiederum ist ein großer Pluspunkt für Grsecurity, allerdings wird dafür der IPTables Patch benötigt!

6.4 Portsentry als Abwehrmaßnahme gegen Portscans

Portsentry ist ein Werkzeug zum Überwachen von Ports auf bestimmte Portscanaktivitäten und kann unter [33] bezogen werden. Portsentry bietet 6 verschiedene Überwachungsfunktionen und kann mittels einer Konfigurationsdatei den eigenen Bedürfnissen angepasst werden.
Die Konfiguration besteht aus insgesamt 11 verschiedenen Abschnitten. Die Wichtigsten lauten hierbei:

- Portkonfiguration
- Erweiterte Stealth Scan Erkennungsmethoden
- Konfigurationsdateien
- Optionen zum Ignorieren von Scanvorgängen
- Optionen zum Beenden von Verbindungen
- Unterstützung von TCP-Wrappern
- Reaktionsoption bei Scanvorgängen
- Ausgabe von Bannern für angegriffene Ports

In der Portkonfiguration wird festgelegt, welche Ports überwacht werden sollen. Diese Ports werden dann von Portsentry in den normalen Überwachungsmethoden berücksichtigt und auf Anomalien überprüft. Mit Hilfe der Reaktionsoptionen bei Scanvorgängen kann man einstellen, auf wievielen Ports man den Zugriff erlauben will, bis ein Angriff gemeldet wird. Dieser Wert, scann_trigger genannt, sollte auf den Default Wert 0 gesetzt bleiben, sodass sofort auf einen falschen Zugriff reagiert wird. Mit Hilfe der Optionen zum Ignorieren von Scanvorgängen und zum Beenden von Verbindungen kann man festlegen, wie auf ein Scanvorgang reagiert werden soll. Defaultmäßig werden TCP- und UDP-Scans geblockt. Bei einem geblockten Zugriffsversuch zeigt Portsentry dann ein entsprechendes Banner an, welches im Abschnitt Ausgabe von Bannern für angegriffene Ports konfiguriert wird. Eine Verbindung von einem Angreifer kann mittels der Option zum Beenden von Verbindungen geblockt werden.
Dafür greift Portsentry unter Linux auf die Filtersoftware IPTables zurück und verbietet Verbindungen mit dem Befehls "iptables -I INPUT -s $TARGET$ -j DROP“.
Neben den Standardüberwachungsmethoden für UDP und TCP kann Portsentry auch so genannte Stealth Scans wie FIN-, X-MAS-, NULL- oder SYN-Scans erkennen. Die entsprechend zu überwachenden Ports werden im Abschnitt „ Erweiterte Stealth Scan Erkennungsmethoden“ konfiguriert.

Erkannte Scanvorgänge werden entsprechend geblockt und können ebenfalls in die host.deny Datei eingetragen werden, sodass TCP-Wrapper darauf zugreifen können und den Zugriff ebenfalls für den angreifenden Host verwehren.

6.5 Firewallsoftware zum Schutz gegen IP-,TCP- und UDP-Angriffen

Vorüberlegungen

Um den Server vor fremden und ungewollten Zugriffen zu schützen, sollte generell eine Firewall eingesetzt werden. Im Rahmen dieser Arbeit soll erläutert werden, wie man kostengünstig eine Firewall unter Linux auf Basis eines statischen Paketfilters aufbaut und wartet.
Dazu wird unter Linux die Software IPTables eingesetzt, die das Filtern der ein- und ausgehenden Pakete übernimmt.
Die Firewall wird aus einem einzelnen Shell-Script bestehen, in denen die Anweisungen für den Paketfilter IPTables stehen. Die Firewall, die hier besprochen wird, befindet sich seit mehreren Monaten in Betrieb auf dem Server zoozle.net. Diese Firewall funktioniert sehr gut und kann falsche Zugriffe und Spoofing erfolgreich abwehren. Der zu schützende Server soll ein Server sein, wie man ihn von Strato oder 1&1 mieten kann.
Darüber hinaus sollen die elementaren Dienste Mail, FTP, Webserver und Secure Shell zugänglich sein, alle anderen Dienste sollen deaktiviert sein, um keine unnötige Angriffsfläche zu bieten.

Das Firewall-Script

Das Firewallscript besteht unter anderem aus folgenden Funktionen firewall_up(), ROC_TWEAKS (), BASIC_PROTECTION(), BANNED_IPS(), TOS_TWEAKS() und firewall_down(). Die Funktion firewall_up bewirkt, dass die Firewall initialisiert und gestartet wird. Wichtige Variablen wie IP-Adressen und Basis- und erweiterte Schutzfunktionen werden hier aktiviert. Weiterhin kann man in dieser Funktion einstellen, welche Art von Diensten aktiviert bzw. deaktiviert werden sollen. Darüber hinaus werden auch die Standardeinstellungen für das Behandeln von ein- und ausgehenden Paketen definiert, die hier zum Schutze der Sicherheit erst einmal alle auf "nicht zulassen" gestellt werden und später dann einzeln und separat aktiviert werden. Die Funktion PROC_TWEAKS ist die elementare Funktion, in der einzelne Schutzmechanismen aktiviert werden. Wichtig hierbei ist der Schutz vor IP-Spoofing, SYN-Flooding, Deaktivierung von ICMP-Braodcasts zur Vermeidung

von IMCP-Flooding und smurf und Fraggle Angriffen und die Aktivierung der "Immer defragmentieren"-Funktion zum Schutz vor böswillig defragmentierten Paketen. Man sieht also, dass die Funktion PROC_TWEAKS ein Großteil der Sicherheitsaspekte zum Schutz des Servers ausmacht und sie wird deshalb standardmäßig aktiviert. Eine weitere wichtige Grundfunktion zur Basissicherheit des Severs stellt BASIC_PROTECTION() dar. Hier werden illegale Pakete abgeblockt, Portscanns werden nicht zugelassen und IP-Pakete, die von unmöglichen Adressen kommen können (z.B. 127.0.0.1) werden hier abgeblockt. Die Funktion BANNDED_IPS liest aus einer bestehenden Datei IP-Adressen ein und blockt Zugriffe von diesen Adressen dann an der Firewall ab. In der Funktion TOS_TWEAKS werden verschiedene Dienste und verschiedene Typ of Service Angaben für das IP-Protkoll gemacht, die unter anderem sicherstellen, dass bestimmte Dienstanforderungen an denjeweiligen Dienst auf dem Server erfüllt werden. Zum Beispiel wird für den Dienst Webserver der Typ of Service Wert 8 gesetzt, was bedeutet, dass man hier den maximalen Durchsatz auf das IP-Protokoll anfordert. Die Funktion firewall_down setzt den Rechner wieder in den ungeschützten Anfangszustand zurück und deaktivert alle Filterregeln. Das komplette Firewallscript wird der Arbeit beigelegt und im Quellenverzeichnis aufgeführt.
Es soll darauf hingewiesen werden, dass man mit dem beigelegten Programm portcloser.pl sehr schnell und einfach definierte Ports unter Linux schließen und öffnen kann.

6.6 Verteidigung gegen SYN-Flooding Angriffe

SYN-Flooding Angriffe sind in den letzten Jahren immer wieder genutzt worden, um einzelne Rechner softwareseitig vom Netzwerk zu trennen. Dies geschieht, indem man den betroffenen Host mit einer unnatürlich hohen Anzahl von TCP-Paketen mit gesetztem SYN-Flag bombardiert. Die kernelinterne Backlog Queue wird mit Verbindungsanfragen überflutet und kann keine neuen Verbindungen mehr speichern – der Dienst ist nicht mehr erreichbar. Dies ist besonders kritisch, wenn eine Maschine mittels SSH gewartet werden muss. Sollte diese nicht mehr erreichbar sein, kann das zum Ausfall des gesamten Rechners führen, was mit Kosten verbunden ist. Aus diesem Grund wurde das Programm anti_syn_flood.pl geschrieben. Es loggt die Anzahl der SYN-Verbindungswünsche von und zum eigenen Host mit. Wenn eine kritische Anzahl an Verbindungsanfragen pro IP überschritten werden, greift dieses Programm ein und generiert ein TCP-Paket, mit gesetztem FIN-Flag und beendet die schädliche Verbindung somit. Das Programm ist im Quellcodeanhang gelistet.

6.7 Schutz vor webbasierenden Angriffen mit ModSecurity

Die Software ModSecurity kann unter modsecurity.org bezogen werden und versteht sich als webbasierende Firewall. ModSecurity wird unter Linux als Apache Module in den Webserver eingebunden und bietet ein breites Spektrum an Verteidungsmöglichkeiten gegnüber web-basierenden Angriffen. Nach der Installation des Apache Moduls muss die Konfigurationsdatei httpd.conf des Webservers Apache angepasst werden.
Dazu fügt man den folgenden Abschnitt hinzu:

```
<IfModule mod_security.c>
    # ModSecurity Filter anschalten
    SecFilterEngine On
    SecAuditEngine RelevantOnly
    # ModSecurity Logfile
    SecAuditLog logs/audit_log
    # Test von POST Content
    SecFilterScanPOST On
    # Default Action
    SecFilterDefaultAction "deny,log,status:406"
    # Schutz vor Pfadangriffen
    SecFilter "\.\./"
    # Schutz vor Javascript Injection
    SecFilter "<(.|\n)+>"
    # Einfacher Schutz gegnüber SQL Injection
    SecFilter "delete[[:space:]]+from"  "deny,status:500"
    SecFilter "insert[[:space:]]+into"  "deny,status:500"
    SecFilter "select.+from"  "deny,status:500"
    #HTTP Anfragen müssen HTTP_USER_AGENT und HTTP_HOST haben
    SecFilterSelective "HTTP_USER_AGENT|HTTP_HOST" "^$"
    # Forbid file upload
    SecFilterSelective "HTTP_CONTENT_TYPE" multipart/form-data
    # Absicherung der Webserverumgebung mit Chroot
    SecChrootPath /chroot/web/apache
</IfModule>
```

Insgesamt betrachtet, gibt es eine Vielzahl von Features, die ModSecurity unterstützt:

- Schutz vor überlangen Variablen
- Filterung von Metazeichen
- Schutz vor SQL Injection Attacks
- Schutz vor Pfadangriffen
- Schutz vor Javascript Injection
- Absicherung der Webserverumgebung mit Chroot

Im Folgenden soll nun detaillierter auf die einzelnen Punkte eingegangen werden. Dazu wird angenommen, dass das PHP-Script secure.php im Verzeichnis sec auf einem Server mit dem Namen server unter der Topleveldomain com besonders geschützt werden soll.
Der Schutz vor überlangen Variablen ist insofern wichtig, da Webserver bei der Verarbeitung von überlangen Daten oft abbrechen und auch unkontrolliert reagieren können. Viele Viren und Würmer im Internet senden oft extrem lange Variablen an den Webserver, um durch eine unkontrollierte Reaktion Zugriff auf den Server zu bekommen. Um die angesprochene Datei secure.php und die POST-Variable string gegenüber diesen Angriffen zu schützen, wird folgende Direktive der Webserverkonfiguration unter dem Abschnitt "<IfModule mod_security.c>" hinzugefügt:

```
<Location /sec/secure.php>
SecFilterInheritance Off
SecFilterDefaultAction "deny,log,status:500"
SecFilterScanPOST On
SecFilterCheckURLEncoding On
SecFilterCheckUnicodeEncoding On
SecFilterSelective POST_PAYLOAD "<\s*string[^>]*>" chain
SecFilterSelective POST_PAYLOAD "<\s*string[^>]*>.{1024,}
</\s*string\s*>" "deny,status:500"
</Location>
```

Bei diesem Beispiel ist die maximal zulässige Anzahl 1024 Bytes für den Umfang der Variable string. Größere Werte führen zur einer Fehlermeldung und werden nicht verarbeitet. Wenn man den eigenen Webserver nun noch gegen schädliche Metazeichen schützen will, kann man obiges Beispiel um folgenden Eintrag erweitern, sodass wiederum die Variable string geschützt wird:

```
SecFilterSelective POST_PAYLOAD "<\s*string[^>]*>.+['\"%][^<]
*</\s*string\s*>" "deny,status:500"
```

Möchte man diese spezielle Variable string nun auch gegen SQL Injection Angriffe schützen, fügt man dem obigen Beispiel noch folgenden Eintrag hinzu:

```
SecFilterSelective POST_PAYLOAD "<\s*string[^>]*>.*select.+from
[^<]*</\s*string\s*>" "deny,status:500"
```

Mit den eben beschriebenen Beispielen kann man seinen Webserver sehr gut gegen webbasierende Angriffe absichern, wobei der komplette Source Code von ModSecurity frei verfügbar ist und

ständig weiterentwickelt wird. Die aktuellste Version stammt vom 20.April 2005. Durch eine ständige Weiterentwickelung ist somit die Sicherheit des Webservers beim Einsatz von ModSecurity gesichert.

6.8 Software gegen Schadprogramme

Die drei bereits besprochenen Programme F-Prot, chkrootkit und Rootkit Hunter sollen bei der Implementation des sicheren Linuxrechners zum Tragen kommen, da man mit diesen 3 Programmen das eigenene Linuxsystem sehr leicht auf Schadprogramme wie Viren, Trojaner, Würmer und Rootkits testen kann, wobei gefundene Schädlinge sofort gelöscht und damit unschädlich gemacht werden.
Mit Hilfe des Programms F-Prot kann man unter Linux nach Viren, Trojanern und Würmern suchen. Das Programm ist für den nichtkommerziellen Einsatz kostenlos .Gefundene Schädlinge werden gelöscht und somit unschädlich gemacht.
Rootkit Hunter ist ein Open Source Programm, welches mit dem Zweck entwickelt wurde, Rootkits unter Linux aufzuspüren und zu löschen. Ein kompletter Systemscan dauert circa 10 Minuten und präsentiert anschließend in übersichtlicher Form das beim Scannen mitgeloggte Protokoll.

6.9 Schutz des Linuxsystems mit Bastille Linux

Im Abschnitt 3 wurde bereits auf die Softwaresuite Bastille Linux eingegangen und wie man damit den eigenen Linuxserver absichern kann, darum soll hier nur nochmal darauf verwiesen werden. Zusätzlich zu Bastille Linux sollte man das eigene System auf vorhandene SETUID-Programme testen, da diese oft einen Angriffspunkt bieten.

Befehl zum Auffinden von SETUID-Programmen:

```
find / -perm -004000 -o -perm -002000  -type f -print
```

Mit dem oben genannten Befehl kann man alle vorhandenen SETUID-Programme auflisten lassen und anschließend entscheiden, ob diese Programme gebraucht werden. Sollte sich herausstellen, dass diese Programme unnötig sind, so sollte man diese entsprechend deaktivieren, um somit keine Angriffsfläche zu bieten.

6.10 Libsafe gegen Buffer Overflows

Die Libsafe Bibliothek wurde entwickelt, um unsichere Funktionen gegenüber der Anfälligkeit für Buffer Overflows zu überwachen. Dabei funktioniert Libsafe mit allen vorkompilierten, ausführbaren Programmen und überwacht dabei, ob potentiell anfällige Funktionsaufrufe verwendet werden. Sollte es der Fall sein, dass eine Funktion durch einen Buffer Overflow kompromittiert wird, so wird diese Funktion durch Libsafes Implementation einer sicheren Funktion ersetzt, sodass es nicht zum Buffer Overflow kommen kann. Die Performance geht dabei kaum zurück, man sollte Libsafe in Kombination mit dem Openwall Security Patch, dem Grsecurity Patch oder mit dem Exec Shield also als kostengünstigen Schutz gegen Buffer Overflow Angriffe nutzen und entsprechend einsetzen. Die folgenden Funktionen werden durch Libsafe überwacht:

- strcpy(char *dest, const char *src)
- strcat(char *dest, const char *src)
- getwd(char *buf)
- gets(char *s)
- [vf]scanf(const char *format, ...)
- realpath(char *path, char resolved_path[])
- [v]sprintf(char *str, const char *format, ...)

Libsafe kann unter [31] bezogen werden und steht als Quellcode zur Verfügung. Nach dem Installieren muss eine Variable z.B. LD_PRELOAD mit dem Wert LD_PRELOAD=/lib/libsafe.so.2 gesetzt werden und diese Variable muss anschließend exportiert werden, sodass sie systemweit zur Verfügung steht. Dies geschieht mit dem Befehl "export LD_PRELOAD". Diese Prozedur sollte man in die Benutzer-Startscripte einbinden, damit Libsafe entsprechend zur Verfügung steht.

6.11 Nessus zum Aufdecken von Systemschwachstellen

Um das eigene System anschließend entsprechend zu testen, sollte man Nessus, Nmap und Nemesis einsetzen. Der Scan mit Nmap sollte dann bei dem gepatchten System in vertretbarer Zeit keine sinnvollen Ergebnisse anzeigen. Nessus hingegen sollte eine kleine Schwachstelle im System aufzeigen – Rootlogins mittels SSH sind erlaubt.
Dieses Problem löst man, indem man in der Datei "/etc/ssh/sshd.conf" den Eintrag "PermitRootLogin No" gesetzt hat. Nun darf sich Root nicht direkt einloggen, sondern muss sich

als normaler Benutzer anmelden und dann mittels su Root Rechte erlangen. Sollte ein extremes Bedürfniss nach Systemsicherheit bestehen, so sollte man die in Abschnitt 4 beschriebenen Maßnahmen hier ebenfalls umsetzten. Weiterhin besteht die Möglichkeit, das System mit Bastille Linux entsprechend zu härten.
Neben dem vorgestellten Sicherheitsscanner Nessus, der aus der Open Source Gemeinde stammt, gibt es noch den Internet Security Scanner von Internet Security Systems. Dieses kommerzielle Produkt kann unter iss.net bezogen werden. Der Vorteil bei diesem Produkt ist es, dass entsprechende Vulnerability sofort mit Bekanntwerden in den Scanner integriert werden, sodass man die damit verbundenen Schwachstellen sofort auffinden und schließen kann. Allerdings soll das Produkt um 5000,- Euro kosten. Preise sind auf der Homepage nicht direkt zu finden, diese erfährt man nur auf Nachfrage an den Sales Support. Aufgrund des hohen Preises dürfte die Investition für viele Firmen nicht in Frage kommen.Es beleibt hierbei zu hoffen, dass auf das kostenlose Nessus zurückgegriffen wird, um das eigene System aus Rechnern trotzdem hinreichend auf Schwachstellen zu testen. Da sich diese Möglichkeit hier sogar kostenlos anbietet, sollte man diese Chance entsprechend nutzen, denn nichts ist schlimmer als ein Einbrecher, der Vollzugriff auf das eigene System nur aufgrund eines einzelnen und kleinen Exploits hat, der einen Fehler im Mail-System ausnutzt. Meistens kommen Angreifer über fehlerhafte PHP-Implemantationen über den Webserver auf den eigenen Server, hier sollte also besonders Bedacht angeraten sein.

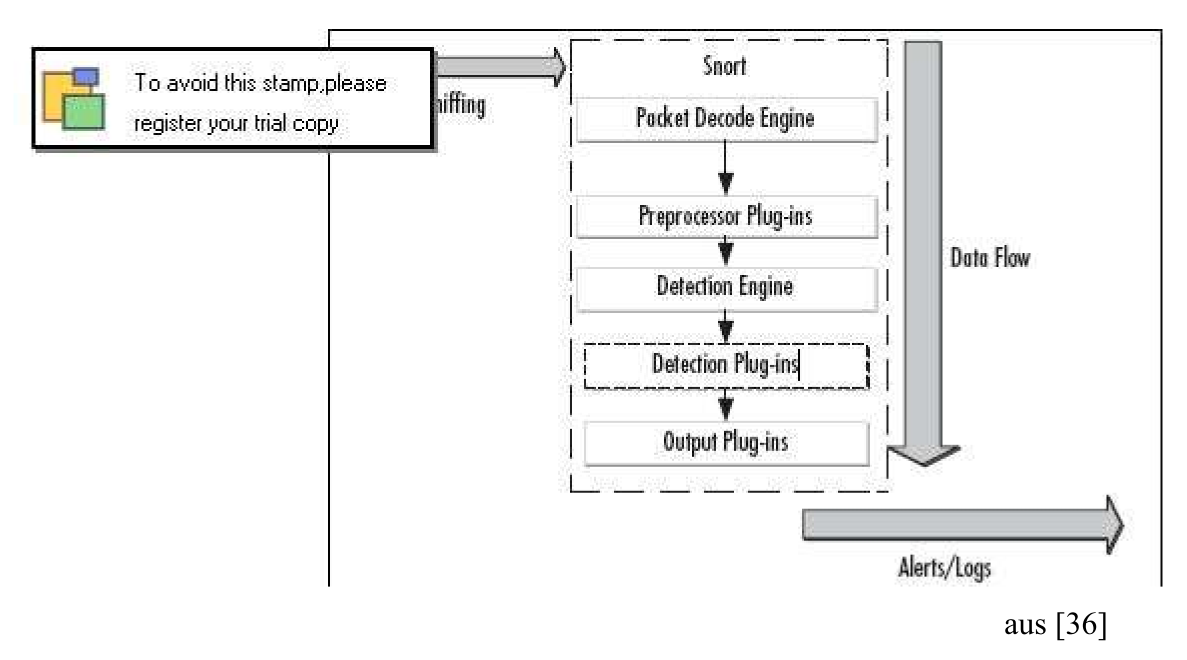
6.12 Angriffserkennung mit Hilfe des Einbruchserkennungssystems Snort

Snort ist ein Einbruchserkennungssystem, das als Open Source vorliegt und unter der Internetadresse snort.org kostenlos bezogen werden kann. Snort besteht in seiner Gesamtheit aus mehreren Quellcodekomponenten, die hier nicht weiter beschrieben werden sollen.
Das Werkzeug Snort besteht aus vier hauptsächlichen Komponenten:

- der Paket Capture und Decoder Einheit
- den Preprozessor Plugins
- der Erkennungs-Einheit
- den Output Plugins

Als erstes wird der Netzwerkverkehr mit Hilfe der libpcap-Bibliothek aufgezeichnet und verarbeitet. Danach werden die ankommenden Pakete durch die Decoder Einheit geschickt, überarbeitet, angepasst und anschließend ausgewertet. Snort unterstützt eine verschiedene Anzahl an Kommu-

nikationsprotokollen, wobei TCP, IP und UDP nur einige davon darstellen.



aus [36]

Abbildung 6.2: Snort Architektur im Überblick

Mit Hilfe der Preprozessor Plugins werden ankommende Pakete genauer betrachtet und eventuell manipuliert. Der Preprozessor entscheidet dann, ob das Paket nur untersucht, modifiziert oder eine Mitteilung darüber gemacht werden soll, um das Paket im Anschluß daran der Detection Einheit zu übergeben. Hier werden die einzelnen Pakete gegen eine Vielzahl von Mustern, die den Snort Regeln entstammen, geprüft. Die Output Plugins verarbeiten daraufhin die entstandenen Daten und geben diese standardmäßig auf dem Terminal aus. [36]

Zusätzlich zum Programm Snort empfiehlt sich der Einsatz von Oinkmaster, ebenfalls ein Open Source Programm, mit dem man die jeweils aktuellsten Snort Signaturen von der Snort Webseite herunterladen kann. Oinkmaster ist unter der Adresse oinkmaster.sourceforge.net zu beziehen. Mit Hilfe des unten genannten Befehles lädt es die aktuellsten Regeln von der Snort Webseite:

```
perl oinkmaster.pl -o /etc/snort/rules
```

Weiterhin nützlich zum Verarbeiten der Snort Logfiles ist das Programm Snortalog. Es analysiert das Snort Logfile und generiert eine Übersicht mit aufgetretenen Angriffen. Das Programm ist ebenfalls von der Snort Webseite zu beziehen.

Möchte man Snort nun als Einbruchserkennungssystem zusätzlich zu dem vorgestellten Firewall Script einsetzen, so muss das Firewall Script um den folgenden Eintrag ergänzt werden:

```
/usr/sbin/snort -D -d -e -k all -v -c /etc/snort/snort.conf -l /
var/log/snort/
```

Danach muss die Firewall mit "firewall reload" neu gestartet werden und Snort wird aktiviert.
Von Zeit zu Zeit sollte dann mit Hilfe von Snortalog eine Zusammenfassung der Ergeignisse erstellt werden.
Snort meldet nun alle verdächtigen Aktionen und protokolliert sie in die Datei alert, die im Verzeichnis "/var/log/snort/" liegt.

6.13 Systemüberwachung mit Hilfe von Syslog

Das Protokollierungswerkzeug Syslog ist das Standardprogramm zum Mitloggen von Systemmeldungen auf Linuxsystemen. Es besitzt Schnittstellen für andere Programme, die dann auf die Protkollierungsfähigkeiten von Syslog zurückgreifen können und aus einer Anwendung heraus Meldungen protokollieren können. Ankommende Meldungen können z.B. vom Kernellogger Klogd oder vom Mailprogramm Sendmail kommen und werden dann mittels Syslog an den Sylogd-Server weitergegeben und anschließend z.B. in eine Ausgabedatei oder auf einem Terminal ausgegeben. Nachrichten, länger als 900 Bytes, werden abgeschnitten und anschließend protokolliert. Mit Hilfe der Konfigurationsdatei "/etc/syslog.conf" kann man bestimmen, welche Art von Nachrichten geloggt werden, und was mit diesen dann geschehen soll. Es ist möglich, die Protokollierung in eine Datei umzuleiten, an einen anderen Host zu senden oder als Mail zu versenden.
Beim Einsatz von Logging-Mechanismen sollte man zusätzlich darauf achten, dass immer genug Speicherplatz für die Log-Dateien zur Verfügung steht und dass andere diese Dateien nicht manipulieren können.
Mit Hilfe des Programms chattr und der unten genannten Anweisung kann man der Datei logfile den Status "nur hinzufügen" erteilen, sodass ein Löschen unmöglich gemacht wird:

```
chattr +a logfile
```

6.14 Vergleich von gesichertem gegenüber ungesichertem Linuxsystem

Anfällig für Angriffsart:	Ungesichertes System	Gesichertes System
IP-Spoofing	X	
IP-Fragmentierungsangriffe	X	
UDP-Spoofing	X	
UDP-Flooding	X	
Broadcast Angriffe	X	
SYN-Flooding	X	
Portscanning	X	
Ausnutzung von Buffer Overflows	X	
Ausnutzung von Heap Overflows	X	
Address Space Bruteforcing	X	
Linking Angriffe	X	
Chroot Angriffe	X	
FIFO Angriffe	X	
Webbasierende Angriffe	X	
Härtungsmethoden		
Softwärehärtung mit Bastille Linux	Möglich	X
Sortwarehärtung des Kernels	Möglich	X
Grsecurity Patch	Möglich	X
IPTabels Patch	Möglich	X
Openwall Security Patch	Möglich	
SE-Linux Patch	Möglich	
Exec Shield Patch	Möglich	Funktionalität vorhanden
ModSecurity als webbasierende Firewall	Möglich	X

Abbildung 6.3: Vergleich: Gesichertes Linuxsystem gegenüber Ungesichertem

7 Zusammenfassung

Die vorliegende Arbeit besteht aus 4 großen Abschnitten: Einleitung, Angriffsarten, Sicherheitsmechanismen und Praktischer Teil der Implementation eines sicheren Linuxrechners.

Im ersten Teil wird die Thematik Sicherheit aufgegriffen und anhand einer Studie gezeigt, wie Firmen das Thema IT-Sicherheit einordnen.Weiterhin wird auf die zugrunde liegenden Täterprofile eingegangen und im Weiteren gezeigt, mit welchen Sicherheitzswerkzeugen man die eigene Sicherheit der Linuxrechner testen kann.
Im Abschnitt „Angriffsarten“ wird detailliert auf die wichtigsten Angriffsarten eingegangen, wobei aufgrund des beschränkten Umfanges der Arbeit nur die wichtigsten Arten aufgezeigt wurden.
Der Abschnitt „Sicherheitsmechanismen“ erläutert, welche allgemeinen Schutzfunktionen es gibt

und wie man diese einsetzen kann, um die Sicherheit des Linuxsystems zu erhöhen.
Im abschließenden Teil „Implementation eines sicheren Linuxrechners“ wird detailliert aufgezeigt, wie man einen Linuxrechner gegen netzwerkbasierende Angriffe schützen kann. Die Implementation fußt auf eine frei verfügbare Software, sodass kaum Kosten für diese entstehen.
Resümierend bleibt festzuhalten, dass mit dieser Arbeit dem Benutzer eine Anleitung zur Absicherung eines Linuxrechers gegeben werden soll, an der er sich orientieren und die vorgestellten Maßnahmen Schritt für Schritt selbst umsetzen kann.

Anhang A – Literaturverzeichnis

[1] Ernst und Young: Global Information Security Survey.
Url: <http://www.ey.com/Global/content.nsf/Germany/Presse_-Pressemitteilungen_2003__Unternehmen_sparen_an_IT-Sicherheit>,
verfügbar am 16.05.2005

[2] Achtert, Werner <w.achtert@tuvit.de>: Bedeutung von Qualität und Sicherheit für den Einsatz in IT-Systemen.
Url: <http://www.software-quality-lab.at/StaticWeb/Download/Infotag%2020030423%20-%20Vortrag%201%20-%20Qualitaet%20und%20Sicherheit.pdf >,
verfügbar am 16.05.2005

[3] Mummert und Company: aboutIT, Informationsmarkt für IT und EDV.
Url: <http://www.aboutit.de/view.php?ziel=/03/32/09.html>,
verfügbar am 16.05.2005

[4] Brauch, Patrick: Distibuted Denial of Service.
In: c't. - Mittweida: Heise Zeitschriften Verlag GmbH & Co. KG. - 14/2004, S. 48

[5] Bundesamt für Sicherheit: Durchführungskonzept für Penetrationstests.
Url: <http://www.bsi.de/literat/studien/pentest/penetrationstest.pdf>,
verfügbar am 16.05.2005

[6] Bundesamt für Sicherheit: Startseite Bundesamt für Sicherheit in der Informationstechnik.
Url: <http://www.bsi.de/>,
verfügbar am 02.08.2005

[7] Kelly Martin <webmaster@securityfocus.com>: SecurityFocus.
Url: <http://www.securityfocus.com/>,
verfügbar am 19.05.2005

[8] CERT <cert@cert.org>: CERT Coordination Center.
Url: <http://www.cert.org/>,
verfügbar am 02.08.2005

[9] Nathan , Jeff <jeff@snort.org>: nemesis.sourceforge.net - Packet injection tool suite.
Url: <http://nemesis.sourceforge.net/>,
verfügbar am 21.04.2005

[10] fyodor <fyodor@insecure.org>: Insecure.Org - Nmap Free Security Scanner, Tools & Hacking resources.
Url: <http://www.insecure.org/>,
verfügbar am 19.05.2005

[11] Autor unbekannt <deraison@nessus.org>: Nessus.
Url: <http://www.nessus.org/>,
verfügbar am 19.05.2005

[12] Crash Override: Internet Allgemeine Schwachstellen und Angriffspunkt.
Url: <http://www.free-it.org/archiv/angriff.html>,
verfügbar am 13.04.2005

[13] Wikipedia: Man-in-the-Middle-Angriffe.
Url: <http://de.wikipedia.org/wiki/Man-In-The-Middle-Angriff>,
verfügbar am 25.05.2005

[14] Klein, Tobias: Linux Sicherheit - Security mit Open-Source-Software - Grundlagen und Praxis. - 1.Aufl. - Heidelberg: dpunkt-Verlag, 2001

[15] Kyas, Othmar; a Camo, Markus: IT-Crackdown - Sicherheit im Internet. - 3.Aufl. - Bonn: mitp-Verlag, 2000

[16] fyodor <fyodor@insecure.org>: Nmap Network Scanner.
Url: <http://www.insecure.org/Nmap/data/Nmap_manpage-de.html>,
verfügbar am 19.05.2005

[17] Wikipedia: Pufferüberlauf.
Url: <http://de.wikipedia.org/wiki/Buffer_Overflow/>,
verfügbar am 14.04.2005

[18] Kalmar, Ralf <info@software-kompetenz.de>: Virtuelles Software Engineering Kompetenzzentrum – Pufferüberläufe.
Url: <http://www.software-kompetenz.de/?22063>,
verfügbar am 14.04.2005

[19] Wikipedia: Computerwurm.
Url: <http://de.wikipedia.org/wiki/Computerw%C3%BCrmer>,
verfügbar am 12.04.2005.

[20] Tietz, Thomas; Ebert, Andreas <andreas@trojaner-info.de>: Trojanische Pferde – Was sind das?
Url: <http://www.trojaner-info.de/beschreibung.shtml>,
verfügbar am 12.04.2005

[21] Wikipedia: Root Kit.
Url: <http://en.wikipedia.org/wiki/Rootkit>,
verfügbar am 13.04.2005

[22] Autor unbekannt: Linux kernel patch from the Openwall Project.
Url: <http://www.openwall.com/linux/README.shtml>,
verfügbar am 14.04.2005

[23] Kelm Stefan; Möller, Klaus <moeller@dfn-cert.de>: DFN-CERT - Informationsbulletin Distributet Denial of Service Angriffe.
Url: <http://www.dfn-cert.de/infoserv/dib/dib-2000-01.html>,
verfügbar am 23.05.2005

[24] Erickson, Jon: Forbidden Code - Jedes System ist zu knacken. - 1.Aufl. - Bonn: mitp-Verlag, 2004

[25] Schneider, Bruce: Secret & Lies - IT-Sicherheit in einer vernetzten Welt. - 1. Aufl. - Heidelberg, 2001

[26] Seitner, Florian: Sicherheitsanker – Zusätzliche Sicherheitskonzepte im Linux-Kernel. In: Linux Magazin. - Mittweida: Linux New Media AG. - 03/2005, S.60-64

[27] PaX: Information From Answers.com.
Url: <http://www.answers.com/topic/pax-4>,
verfügbar am 26.06.2005

[28] Spengler, Brad <spender@grsecurity.net>: grsecurity.
Url: <http://www.grsecurity.net/>,
verfügbar am 26.06.2005.

[29] Autor unbekannt: Linux kernel patch from the Openwall Project.
Url: <http://www.openwall.com/linux/>,
verfügbar am 27.06.2005

[30] Quaritsch, Markus; Winkler, Thomas <tom@qwws.net>: Linux - Ein Einblick in den Kernel.
Url: <http://uni.qwws.net/linpro/data/docu/seminar_20040318.pdf>,
verfügbar am 27.06.2005

[31] Fox, Micheal; Giordana, John; Stotler, Lori; Arun, Thomas <at4acs.virginia.edu>: SE-Linux and Grsecurity – "A Side-by-Side Comparison of Mandatory Access Control and Access Control List Implementations".
Url: <http://it97.dyndns.org/Security/SELinuxGRsecuritySElinux.pdf>,
verfügbar am 27.06.2005

[32] Autor unbekannt <libsafe@research.avayalabs.com>: Libsafe Manpage.
Url: <http://www.research.avayalabs.com/project/libsafe/doc/libsafe.8.html >,
verfügbar am 31.05.2005

[33] H. Rowland, Craig <craigrowland@users.sourceforge.net >: SourceForge.net: Project Info - Sentry Tools.
Url: <http://sourceforge.net/projects/sentrytools/>,
verfügbar am 05.07.2005

[34] Autor unbekannt: Der TCP-Wrapper.
Url: <http://www.linuxfibel.de/wrapper.htm>,
verfügbar am 05.07.2005

[35] Shah, Shreeraj: Infosecwriters.com – Defending-web-services.pdf.
Url: <http://www.infosecwriters.com/text_resources/pdf/Defending-web-services.pdf>,
verfügbar am 06.07.2005

[36] Beale, Jay; C. Foster, James; Posluns, Jeffrey ... : Snort 2 Intrusion Decection. - 1.Aufl. - USA: Syngress Publishing Verlag, 2003

[37] Venema, Dr. W.Z. <wietse@porcupine.org >: Wietse's collection of tools and papers.
Url: <http://ftp.porcupine.org/pub/security/tcp_wrappers_7.6.tar.gz>,
verfügbar am 03.08.2005

[38] Autor unbekannt: IPTables Firewall.
Url: <http://www.free-it.org/fw/>,
verfügbar am 17.04.2005

Anhang B – Programmverzeichnis

Datei nemesis_icmp_ping.pl:

```
#!/usr/bin/perl

# Autor: Sebastian Enger
# Bachelorarbeit: Implementation eines sicheren Linuxrechners
# Datum: 20.5.2005
# Zweck: Erstellung von ICMP Paketen mittels Nemesis, Testzwecke->ICMP
Nachrichten versenden

use Getopt::Std;
%options=();
getopts("a:b:c:d:f:e:h",\%options);

if ( defined $options{h}) {
         &usage();
}

main_routine();

sub main_routine() {

# Parameter von getopts entgegennehmen
$source_ip = $options{a};$source_dst_ip = $options{b};
$dest_ip  = $options{c};$dest_dst_ip   = $options{d};
$payload_file = $options{f};$packet_count      = $options{e};

for ($i=0;$i<=$packet_count;$i++) {
         # Paket mittels Nemesis assemblieren und versenden
         system("nemesis icmp -qE -B $source_ip -b $source_dst_ip -D $dest_ip
-S $dest_dst_ip -I 4 -T 250 -P $payload_file"); }

} # main_routine();

sub usage() {

print STDERR << "EOF";
This program creates and sends out a icmp echo request.

    usage: $0 [-habcd] [-f file]

     -h          : this (help) message
     -a           : orignial source ip address
     -b           : orignial destination ip address
```

```
     -c            : source ip address
     -d            : destination ip adress
     -f file    : file containing tcp payload of the packet
     -e count      : count paketes are generated and send

example: $0 -a 141.55.123.24 -b 151.23.45.67 -c 141.55.123.24 -d 151.23.45.67 -f
payload.txt

EOF
        exit;
}
```

Datei nemesis_udp_flood.pl:

```
#!/usr/bin/perl
# Autor: Sebastian Enger
# Bachelorarbeit: Implementation eines sicheren Linuxrechners
# Datum: 20.5.2005
# Zweck: Erstellung von UDP Paketen mittels Nemesis, Testzwecke->UDP Flooding

use Getopt::Std;
%options=();
getopts("a:b:c:d:f:e:h",\%options);

if ( defined $options{h}) {
         &usage();
}

main_routine();

sub main_routine() {

# Parameter von getopts entgegennehmen
$source_ip         = $options{a};
$source_port       = $options{b};
$dest_ip  = $options{c};
$dest_port         = $options{d};
$payload_file      = $options{f};
$packet_count      = $options{e};

 # Erstellung der Pakte mit der von $paket_count vorgegebenen Anzahl->UDP Paket
for ($i=0;$i<=$packet_count;$i++) {
         # Paket mittels Nemesis assemblieren und versenden
         system("nemesis udp -S $source_ip -x $source_port -D $dest_ip -y
$dest_port -I 4 -T 250 -P $payload_file");
}

} # main_routine();

sub usage() {

print STDERR << "EOF";

    This program creates and sends UDP storms

    usage: $0 [-habcd] [-f file]

     -h           : this (help) message
     -a            : source ip address
     -b            : source port
     -c            : destination ip address
```

```
     -d            : destination port
     -f file    : file containing tcp payload of the packet
     -e count      : count paketes are generated and send

example: $0 -a 141.55.123.24 -b 22 -c 151.23.45.67 -d 22 -f payload.txt

EOF
        exit;
}
```

Datei nemesis_syn_flood.pl:

```
#!/usr/bin/perl

# Autor: Sebastian Enger
# Bachelorarbeit: Implementation eines sicheren Linuxrechners
# Datum: 20.5.2005
# Zweck: Erstellung von TCP Paketen mit gesetztem SYN-Flag mittels Nemesis,
Testzwecke->SYN Flooding

use Getopt::Std;
%options=();
getopts("a:b:c:d:f:e:h",\%options);

if ( defined $options{h}) {
         &usage();
}

main_routine();

sub main_routine() {

# Parameter von getopts entgegennehmen
$source_ip         = $options{c};
$source_port       = $options{b};
$dest_ip  = $options{a};
$dest_port         = $options{d};
$payload_file      = $options{f};
$packet_count      = $options{e};

 # Erstellung der Pakte mit der von $paket_count vorgegebenen Anzahl->TCP Paket
mit gesetztem SYN-Flag
for ($i=0;$i<=$packet_count;$i++) {
         # Paket mittels Nemesis assemblieren und versenden
         system("nemesis tcp -S $source_ip -x $source_port -D $dest_ip -y
$dest_port -fS -I 4 -T 250 -P $payload_file");
}

} # main_routine();

sub usage() {

print STDERR << "EOF";

This program creates and sends tcp syn storms

    usage: $0 [-habcd] [-f file]

     -h           : this (help) message
     -c            : source ip address
     -b            : source port
```

```
     -a            : destination ip address
     -d            : destination port
     -f file    : file containing tcp payload of the packet
     -e count      : count paketes are generated and send

example: $0 -a 141.55.123.24 -b 22 -c 151.23.45.67 -d 22 -f payload.txt

EOF
        exit;
}
```

Datei genpasswd.pl:

```
#!/usr/bin/perl

# Autor: Sebastian Enger
# Bachelorarbeit: Implementation eines sicheren Linuxrechners
# Datum: 20.5.2005
# Zweck: Erstellung von sicheren Passwörtern

# Umfang des Zeichensatzes für Passwort festlegen
$numbers            = "1234567890";
$alpha_small        = "abcdefghijklmnopqrstuvwxyz";
$alpha_big          = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
$extra              = "!@#$%^&*()_+-=[]{}\|;':\",./<>?`";

# String in Array umwandeln
@numbers            = split(//,$numbers);
@alpha_small        = split(//,$alpha_small);
@alpha_big          = split(//,$alpha_big);
@extra              = split(//,$extra);

# Programmroutine aufrufen
&main_routine;

sub main_routine {

# Variable mit Wert aus Zufallsgenerator auffüllen
my @rand_id_num = rand_choice_number();
my @rand_id_sma = rand_choice_small();
my @rand_id_big = rand_choice_big();
my @rand_id_ext = rand_choice_extra();

# Zuordnung Zufallszahl zu Umfang des Zeichensatzes für Passwort
foreach $num (@rand_id_big) {
         push(@passwd_raw,@alpha_big[$num]);
}

foreach $num (@rand_id_num) {
         push(@passwd_raw,@numbers[$num]);
}

foreach $num (@rand_id_sma) {
         push(@passwd_raw,@alpha_small[$num]);
}

foreach $num (@rand_id_ext) {
         push(@passwd_raw,@extra[$num]);
}

# Passwortstring zusammensetzen
$passwd =  @passwd_raw[0] . @passwd_raw[1] . @passwd_raw[2] . @passwd_raw[3] .
```

```
@passwd_raw[4] .@passwd_raw[5] . @passwd_raw[6] .@passwd_raw[7] . @passwd_raw[8]
. @passwd_raw[9]. @passwd_raw[10];

# Ausgabe
print "$passwd\n";

sub rand_choice_extra {

         @random_numbers1 = ();

         $rand_choice_1 = int( rand(31)) + 0;
         $rand_choice_2 = int( rand(31)) + 0;
         $rand_choice_3 = int( rand(31)) + 0;

         push(@random_numbers1,$rand_choice_1);
         push(@random_numbers1,$rand_choice_2);
         push(@random_numbers1,$rand_choice_3);
         return @random_numbers1;

}

sub rand_choice_small {

         @random_numbers2 = ();
         $rand_choice_1 = int( rand(26)) + 0;
         $rand_choice_2 = int( rand(26)) + 0;
         $rand_choice_3 = int( rand(26)) + 0;

         push(@random_numbers2,$rand_choice_1);
         push(@random_numbers2,$rand_choice_2);
         push(@random_numbers2,$rand_choice_3);
         return @random_numbers2;

}

sub rand_choice_big {

         @random_numbers3 = ();

         $rand_choice_1 = int( rand(26)) + 0;
         $rand_choice_2 = int( rand(26)) + 0;
         $rand_choice_3 = int( rand(26)) + 0;

         push(@random_numbers3,$rand_choice_1);
         push(@random_numbers3,$rand_choice_2);
         push(@random_numbers3,$rand_choice_3);
         return @random_numbers3;

}

sub rand_choice_number {

         @random_numbers4 = ();

         $rand_choice_1 = int( rand(10)) + 0;
         $rand_choice_2 = int( rand(10)) + 0;
         $rand_choice_3 = int( rand(10)) + 0;

         push(@random_numbers4,$rand_choice_1);
         push(@random_numbers4,$rand_choice_2);
         push(@random_numbers4,$rand_choice_3);
         return @random_numbers4;

}

} #main_routine
```

Datei support_genpasswd.pl:

```
#!/usr/bin/perl

# Autor: Sebastian Enger
# Bachelorarbeit: Implementation eines sicheren Linuxrechners
# Datum: 20.5.2005
# Zweck: Zusatztool zum Erstellung von sicheren Passwörtern
# Anzahl der zu erstellenden Passwörter von Kommandozeile entgegennehmen
$count = $ARGV[0];
# Programm genpasswd.pl aufrufen, so oft wie $count vorgibt
for ( my $i = 0; $i <= $count; $i++) { system("perl genpasswd.pl"); }
```

Datei portcloser.pl:

```
#!/usr/bin/perl

# Autor: Sebastian Enger
# Bachelorarbeit: Implementation eines sicheren Linuxrechners
# Datum: 20.5.2005
# Zweck: Sofortiges Schließen und Öffnen von Ports

# Übergebene Parameter einlesen
$ARG = $ARGV[0];

# Sicherheitscheck, $0 läuft nur als Root
$who =`whoami`;

if ($who !~ /root/){
         print "sorry, need rOOt !\n";
         exit;
}

# folgende Dienste werden unterstützt:
@services = ("vsftp","atd","apache","squid","smail","portmap","samba","rsync");

# wenn Argument 'start' lautet, starte die Dienste, sonst schließe sie
if ($ARG == "start"){
         foreach $service (@services) {
                  print "STARTING: $service\n";
                  &start_services_now($service);
         };
} elsif($ARG == "stop"){
         foreach $service (@services) {
                  print "STOPPING: $service\n";
                  &stop_services_now($service);
         };
# sonst immer Dienste beenden
} else {
         print "Secure Modus on!Shutting down services "
         foreach $service (@services) {
                  print "STOPPING: $service\n";
                  &stop_services_now($service);
         };
};

sub start_services_now {
         my $service = @_[0];
         system("/etc/init.d/$service start");
```

```
};

sub stop_services_now {
         my $service = @_[0];
         system("/etc/init.d/$service stop");
};
```

Datei anti_syn_flood.pl:

```
#!/usr/bin/perl

# Autor: Sebastian Enger
# Bachelorarbeit: Implementation eines sicheren Linuxrechners
# Datum: 20.5.2005
# Zweck: Verteidung gegen SYN-Flood

# Module laden
use Net::RawIP qw(:pcap);

# Bildschirm löschen
system("clear");

# Signalhandler installieren
$SIG{INT} = \&quit;

# Variablen vordefinieren
my ($ip,$seq_nr,$count);

# Meine IP festlegen->anpassen!
$my_ip = "123.123.123.123";
$count = 0;

# Autoflush an
$|=1;

$dev='eth0';
$ip_addr=${ifaddrlist()}{$dev};

# Initialisierung
$packet_tcp=new Net::RawIP({tcp=>{}});
$filt_tcp="ip proto \\tcp";
$pcap_tcp=$packet_tcp->pcapinit($dev,$filt_tcp,1500,60);
$offset_tcp = linkoffset($pcap_tcp);

if (fork){ loop $pcap_tcp,-1,\&check_tcp,\@packet_tcp;}

sub check_tcp{

my $time = timem();
$packet_tcp->bset($_[2],$offset_tcp);
$proto=$packet_tcp->proto;

my
($saddr,$daddr,$sport,$dport,$seq,$ack_seq,$urg,$ack,$psh,$rst,$syn,$fin,$data)=
$packet_tcp->get({ip=>['saddr','daddr'],tcp=>
['source','dest','seq','ack_seq','urg','ack','psh','rst','syn','fin','data']});

if ($sport == "22" ) {

# vorerst ssh ignorieren
} elsif ( $dport == "22" ) {
} else {
```

```
#$seq =~ s/-//g;
# nur wenn SYN Paket ankommt
if ($syn){         # & !$ack) {

print "PACKET: $seq ";
if ($urg) {print "URG "};
if ($ack) {print "ACK "};
if ($psh) {print "PSH "};
if ($rst) {print "RST "};
if ($syn) {print "SYN "};
if ($fin) {print "FIN "};

if ($saddr > $daddr) {

# von unserer IP zu fremder IP
print "\t",ip2name($saddr), ":$sport \t -> \t",ip2name($daddr),":$dport \n";
push(@ips, ip2name($daddr) . "#$seq");

} else {

# von fremder IP zu uns
print "\t",ip2name($daddr), ":$dport \t <- \t",ip2name($saddr),":$sport \n";
# ATTACKER_IP - SEQ - ATTACK_PORT - MY_PORT
# @content enthält die Information über die letzten 120 Verbindungsanfragen
push(@content, ip2name($saddr) . "#$seq#$sport#$dport");
# bei mehr als 120 Einträgen im Array @content

if ($#content >= "120"){

foreach $entry(@content){
($ip,$seq_nr,$attacker_port,$my_port) = split('#', $entry);

if ($ip == ip2name($saddr) ){
$count++;
}

# wenn mehr als 100 SYN Anfragen von einem Host ankommen->SYN Flood entdeckt
if ($count >= "100"){
print "INCOMING SYNFLOOD DETECTED!\n";
$attacker = $ip;
foreach $bad (@content) {
($ipX,$seq_nrX,$attacker_portX,$my_portX) = split('#', $bad);

if ($ipX == $attacker){
print "KILLING BAD CONNECTION ATTEMPT!\n";
# generiere ein TCP Paket mit gesetzem FIN Flag,um Verbindung zu beenden
                                                                              my
$new_packet= new Net::RawIP;
$new_packet->set({
ip=>{
         saddr=>$my_ip,
         daddr=>$attacker
},

tcp=>{
         source=>$my_port,
         dest=>$attacker_portX,
         fin=>"1",
         seq=>$seq_nrX
}
});

$new_packet->send;
}}}}
$count = 0;
@content = ();
```

```
}}}}}

sub ip2name {
my $addr = shift;
(gethostbyaddr(pack("N",$addr),AF_INET))[0] || ip2dot($addr);
}

sub ip2dot {
sprintf("%u.%u.%u.%u",unpack "C4", pack "N1", shift);
}

sub quit {
exit(0);
}
```

Datei debian_patch_download.pl:

```
#!/usr/bin/perl

# Autor: Sebastian Enger
# Bachelorarbeit: Implementation eines sicheren Linuxrechners
# Datum: 20.5.2005
# Zweck: Check nach Security Patches für Debian-Systeme und lädt diese bei
Bedarf

use LWP::Simple;
use HTML::LinkExtor;
use LWP::UserAgent;
use URI::URL;

# Hash mit Zuordnung Jahr -> Downloadseite
%debian_security_sites = (

       "2005" => 'http://www.debian.org/security/2005/',
       "2004" => 'http://www.debian.org/security/2004/',
       "2003" => 'http://www.debian.org/security/2003/',
       "2002" => 'http://www.debian.org/security/2002/',
       "2001" => 'http://www.debian.org/security/2001/',
       "2000" => 'http://www.debian.org/security/2000/',
   );


$count = "1";

system("clear");
print "$0 - check for Security Patches for Debian Linux and download them\n";
print "Enter Year to check for updates [2000 2001 2002 2003 2004 2005]\n";

# Benutzereingabe entgegennehmen
chomp($year=<STDIN>);

# bei fehlerhafter Eingabe: beenden
if ($year !~ /\d/) {
         print "Only Numbers please!\n";
         print "Exiting now!\n";
         exit;

# bei fehlerfreier Eingabe: starten
} else {

# Ordnerstruktur für Download erstellen
# /tmp/security_patches/$year/$number -> download
```

```
mkdir("/tmp/security_patches", 0755);
mkdir("/tmp/security_patches/$year", 0755);

# für jedes Hash Paar, splitte in Key und Wert
while ( my ($key, $value) = each(%debian_security_sites) ) {

# wenn Key gleich der Benutzereingabe für Jahr ist, fahre fort :-)
if ($key == $year) {
# benutze die Funktion get() aus dem LWP::Simple Modul, um Seite zu laden
$doc = get($value);

# es folgt der Algorithmus zum Extrahieren der wichtigen Information z.B.:
fix_kde.debs
@content = split(/\n/, $doc);
foreach $line (@content){

# es wurde ein Security Patch gefunden
if ($line =~ /<tt>/) {

# hole Eintrag
@important = split(/>/,$line);
@important[1] =~ s/<\/tt//;
@it = split(/"/,@important[3]);
substr(@it[1], 0, 2) = "";
@important[4] =~ s/<\/a//;
@important[6] =~ s/ - //;
@important[6] =~ s/ <br//;

print "DATE: @important[1]\n";
print "PROG: @important[4]\n";
print "KIND: @important[6]\n\n";
print " URL: ($count) $value@it[1]\n";         # I LOVE PERL!

# Daten in Array schieben
push(@download, "$count $value@it[1]");
$count++;
}}}}

print "\nDownload which Security Advisory?Type number or all to   download all
to /tmp!\n";
chomp($files = <STDIN>);

if ($files =~ /all/i){

print "DOWNLOADING ALL PATCHES FROM $year to /tmp/security_patches/$year/ !\n";
sleep 1;

# für jeden Eintrag: lade Patch herunter
foreach $f_entry (@download){

# split Number und Uri
@uris = split(/ /,$f_entry);

# Download der Daten erfolgt nun
&gather_patchfiles(@uris[0],@uris[1]);
}

} else {

print "DOWNLOADING CHOOSEN FILES TO /tmp/security_patches/$year/ ";
@files = split(/ /,$files);
foreach $entry (@files){

foreach $number (@download){
if ($entry == $number){
```

```
@imp = split(/ /, $number);
&gather_patchfiles(@imp[0],@imp[1]);
}}}}}

sub gather_patchfiles {

$patch_number      = @_[0];
$url               = @_[1];

$ua = new LWP::UserAgent;
$p = HTML::LinkExtor->new(\&collect);

sub collect {

my($tag, %attr) = @_;
if  ($tag =~ m/^a$/i ) {   # Nur <A HREF ...
push(@A_HREF, values %attr);
}}# sub collect

$response = $ua->request(HTTP::Request->new(GET => $url), sub { $p->parse($_[0])
} );

$base = $response->base;
@A_HREF = map { $_ = url($_, $base)->abs; } @A_HREF;

foreach $file (@A_HREF){

# nur *.deb Dateien enthalten Patches
if (reverse($file) =~ /^bed./){

@filename = split(/\//, $file);
print "Downloading @filename[$#filename] to $year/$patch_number\n";
mkdir("/tmp/security_patches/$year/$patch_number", 0755);

# Verzeichniswechsel
chdir("/tmp/security_patches/$year/$patch_number");

# benutze die Funktion getstore() aus dem LWP::Simple Modul, um Patch zu laden
getstore("$file","@filename[$#filename]");
}}}# sub gather_patchfiles
```

Datei firewall.sh:

```
#!/bin/bash

IPT=/sbin/iptables       # Path to iptables

firewall_up() {

#----------------------------#
#        FUNCTIONS           #
#----------------------------#


WANIFACE=eth0              # WAN interface designation (to internet)
LANIFACE=eth0              # LAN interface designation (to hub/switch)
LAN=81.XXX.XXX.XXX         # insert IP here
PROC_TWEAKS=ON             # ON, OFF  Turn OFF only for troubleshooting!
BASIC_PROTECTION=ON        # ON, OFF  Basic protective ruleset.
IANA_RESERVED=OFF          # ON, OFF  Deny all IANA reserved address spaces
ICMP_TRAFFIC=LIMIT         # ON, LIMIT, OFF
TOS_TWEAKS=OFF             # ON, OFF
```

```
SNAT=OFF
MASQ=OFF
BANNED_IPS=/etc/fw_bann      # NONE, PATH  Full path & filename of the text file
                             # containing the banned IP list

# prepare the snort alert logfile and save them to /
#webserver/wwwroot/html/snort.html
/usr/bin/perl /etc/snort/snortalog.pl -report -h /
webserver/wwwroot/html/snort.html -g jpeg -file /var/log/snort/alert

# start Snort as Intrusion Detection System
/usr/sbin/snort -D -d -e -k all -v -c /etc/snort/snort.conf -l /var/log/snort/


#-----------------------------#
#         SERVICES            #
#-----------------------------#

SNMP=OFF
SAMBA=OFF
WEBMIN=OF
FTP=ON              # ON, OFF, IP  File Transfer Protocol
SSH=ON              # ON, OFF, IP  Secure Shell
SMTP=ON             # ON, OFF, IP  Simple Mail Transfer Protocol
DNS=OFF             # ON, OFF, IP  Domain Name Service
TFTP=OFF            # ON, OFF, IP  Trivial File Transfer Protocol
HTTP=ON
AUTH=OFF            # ON, OFF, IP  Authentication Service
POP3=OFF            # ON, OFF, IP  Post Office Protcol version 3
NNTP=OFF            # ON, OFF, IP  Network News Transfer Protocol
NTP=OFF             # ON, OFF, IP  Network Time Protocol
IMAP4=OFF           # ON, OFF, IP  Interim Mail Access Protocol version 4
LDAP=OFF            # ON, OFF, IP  Lightweight Directory Access Protocol
SSL=OFF             # ON, OFF, IP  Secure Sockets Layer
SQUID=OFF           # ON, OFF, IP  Transparent redirect Squid Web Proxy Cache


# If there are other incoming ports which you wish to allow and are not listed
#in the options above, add them to the following variables below.  Ports should
#be seperated by a space unless you wish to specify a range.  To specify a
#range you should seperate the port range with a colon.  Here is an example to
#illustrate:  TCP_PORTS="22 53 80 4000:4100 5000"

TCP_PORTS="OFF"         # OFF, PORTS
UDP_PORTS="OFF"         # OFF, PORTS

#-----------------------------#
#         AUTOCONFIG          #
#-----------------------------#

WANIP=`ifconfig $WANIFACE | grep inet | cut -d : -f 2 | cut -d \  -f 1`
WANMASK=`ifconfig $WANIFACE | grep Mask | cut -d : -f 4`
WANBCAST=`ifconfig $WANIFACE | grep inet | cut -d : -f 3 | cut -d \  -f 1`

LANIP=`ifconfig $LANIFACE | grep inet | cut -d : -f 2 | cut -d \  -f 1`
LANMASK=`ifconfig $LANIFACE | grep Mask | cut -d : -f 4`
LANBCAST=`ifconfig $LANIFACE | grep inet | cut -d : -f 3 | cut -d \  -f 1`

echo "WAN IP:" $WANIP
echo "LAN IP:" $LANIP

################################################################################
###
#                           FLUSH TABLES & SET POLICIES
#
```

```
################################################################################
###

# Flush rules in the filter & nat tables
if [ -f $IPT ]; then
   CHAINS=`cat /proc/net/ip_tables_names 2>/dev/null`
      for TABLE in $CHAINS; do $IPT -t $TABLE -F; done; $IPT -F
      for TABLE in $CHAINS; do $IPT -t $TABLE -X; done; $IPT -X
      for TABLE in $CHAINS; do $IPT -t $TABLE -Z; done; $IPT -Z
   fi

   CHAINS="INPUT FORWARD OUTPUT"
   for TABLE in $CHAINS
   do $IPT -P $TABLE ACCEPT
   done

   CHAINS="PREROUTING POSTROUTING OUTPUT"
   for TABLE in $CHAINS; do $IPT -t nat -P $TABLE ACCEPT; done

   CHAINS="PREROUTING OUTPUT"
   for TABLE in $CHAINS; do $IPT -t mangle -P $TABLE ACCEPT; done
rm -f /var/lock/subsys/iptables

# Set policies to drop in the filter table
$IPT -P INPUT DROP
$IPT -P FORWARD DROP
$IPT -P OUTPUT DROP

# Set policies in the nat table
$IPT -t nat -P PREROUTING ACCEPT
$IPT -t nat -P POSTROUTING ACCEPT
$IPT -t nat -P OUTPUT ACCEPT

# Set policies in the mangle table
$IPT -t mangle -P PREROUTING ACCEPT
$IPT -t mangle -P OUTPUT ACCEPT

################################################################################
###
#                          EXPLOIT PROTECTION & TWEAKS
#
################################################################################
###

#-----------------------------------------#
#            Proc Tweaks                  #
#-----------------------------------------#

function PROC_TWEAKS {

# Disable logging of misc TCP conntracking
if [ -e /proc/sys/net/ipv4/netfilter ]; then
   for x in /proc/sys/net/ipv4/netfilter/ip_ct_tcp_log_*; do
   echo 0 > $x; done
fi

# Disable proxy arp (needed to be enabled for DMZ)
if [ -e /proc/sys/net/ipv4/conf/all/proxy_arp ]; then
   echo 0 > /proc/sys/net/ipv4/conf/all/proxy_arp
fi

# Enable bogus error message protection
if [ -e /proc/sys/net/ipv4/icmp_ignore_bogus_error_responses ]; then
   echo 1 > /proc/sys/net/ipv4/icmp_ignore_bogus_error_responses
fi
```

```
# Enable support for spoof and DOS protection
# Enable TCP SYN Cookie Protection
if [ -e /proc/sys/net/ipv4/tcp_syncookies ]; then
   echo 1 > /proc/sys/net/ipv4/tcp_syncookies
fi

# Enable source address verification to prevent spoofing
if [ -e /proc/sys/net/ipv4/conf/all/rp_filter ]; then
   for x in /proc/sys/net/ipv4/conf/*/rp_filter; do
   echo 1 > $x; done
fi

# Disable source routed packets to help prevent access to LAN
if [ -e /proc/sys/net/ipv4/conf/all/accept_source_route ]; then
   for x in /proc/sys/net/ipv4/conf/*/accept_source_route; do
   echo 0 > $x; done
fi

# Disable TCP Explicit Congestion Notification Support
if [ -e /proc/sys/net/ipv4/tcp_ecn ]; then
   echo 0 > /proc/sys/net/ipv4/tcp_ecn
fi

# Disable ICMP redirects (needed for transparent proxy)
if [ -e /proc/sys/net/ipv4/conf/all/send_redirects ]; then
   echo 0 > /proc/sys/net/ipv4/conf/all/send_redirects
fi

# Disable acceptance of ICMP redirects to avoid malicious routing changes
if [ -e /proc/sys/net/ipv4/conf/$WANIFACE/accept_redirects ]; then
   echo 0 > /proc/sys/net/ipv4/conf/$WANIFACE/accept_redirects
fi
if [ -e /proc/sys/net/ipv4/conf/$WANIFACE/send_redirects ]; then
   echo 0 > /proc/sys/net/ipv4/conf/$WANIFACE/send_redirects
fi

# Accept redirects only from gateways in the default gateways list
if [ -e /proc/sys/net/ipv4/conf/all/secure_redirects ]; then
   echo 1 > /proc/sys/net/ipv4/conf/all/secure_redirects
fi

# Ignore broadcast ICMP echo requests to prevent becoming a Smurf attack
amplifier
if [ -e /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts ]; then
   echo 1 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts
fi

# Drop the ECN flag in tcp-packets
if [ -e /proc/sys/net/ipv4/tcp_ecn ];then
   echo 0 > /proc/sys/net/ipv4/tcp_ecn
fi

# Adjust connection tracking timeout value
# Default=600 (600 seconds or 10 minutes)
if [ -e /proc/sys/net/ipv4/netfilter/ip_ct_generic_timeout ]; then
   echo 120 > /proc/sys/net/ipv4/netfilter/ip_ct_generic_timeout
fi

# Increase maximum limit of connections to track (default=2048)
if [ -f /proc/sys/net/ipv4/ip_conntrack_max ]; then
   echo "13001" > /proc/sys/net/ipv4/ip_conntrack_max
fi

# Disable Log packets from illegal addresses
if [ -f /proc/sys/net/ipv4/conf/all/log_martians ]; then
   echo 0 > /proc/sys/net/ipv4/conf/all/log_martians
```

```
fi

# Enable always defragging Protection
echo 1 > /proc/sys/net/ipv4/ip_always_defrag

# enhance backlog queue for incoming packet to disharm flooding
echo 4096 >> /proc/sys/net/ipv4/tcp_max_syn_backlog

# Enable timestamps
if [ -f /proc/sys/net/ipv4/tcp_timestamps ]; then
   echo 1 > /proc/sys/net/ipv4/tcp_timestamps
fi
}

if [ $PROC_TWEAKS = "ON" ]; then
   PROC_TWEAKS
fi

#-----------------------------------------#
#          Basic Protections              #
#-----------------------------------------#

function BASIC_PROTECTION {

$IPT -N ILLEGAL
$IPT -F ILLEGAL

# Furtive port scanner
$IPT -A ILLEGAL -p tcp --tcp-flags SYN,ACK,FIN,RST RST -m limit --limit 1/s -j
ACCEPT

# Drop illegal flag combinations which also prevents most port scanning
$IPT -A ILLEGAL -i $WANIFACE -p tcp --tcp-flags ALL ALL -j DROP
$IPT -A ILLEGAL -i $WANIFACE -p tcp --tcp-flags ALL NONE -j DROP
$IPT -A ILLEGAL -i $WANIFACE -p tcp --tcp-flags ALL FIN,URG,PSH -j DROP
$IPT -A ILLEGAL -i $WANIFACE -p tcp --tcp-flags ALL SYN,RST,ACK,FIN,URG -j DROP
$IPT -A ILLEGAL -i $WANIFACE -p tcp --tcp-flags SYN,RST SYN,RST -j DROP
$IPT -A ILLEGAL -i $WANIFACE -p tcp --tcp-flags SYN,FIN SYN,FIN -j DROP
$IPT -A ILLEGAL -i $WANIFACE -p tcp --tcp-flags FIN,RST FIN,RST -j DROP
$IPT -A ILLEGAL -i $WANIFACE -p tcp --tcp-flags ACK,FIN FIN -j DROP
$IPT -A ILLEGAL -i $WANIFACE -p tcp --tcp-flags ACK,PSH PSH -j DROP
$IPT -A ILLEGAL -i $WANIFACE -p tcp --tcp-flags ACK,URG URG -j DROP

# Refuse directed broadcasts used in Smurf/Fraggle type DOS attacks
$IPT -A ILLEGAL -i $WANIFACE -d 255.255.255.255 -j DROP
$IPT -A ILLEGAL -i $WANIFACE -d $WANBCAST -j DROP

# Refuse spoofed packets pretending to be from your IP address
$IPT -A ILLEGAL -i $WANIFACE -s $WANIP -d $WANIP -j DROP

# Drop Fragments
$IPT -A ILLEGAL -i $WANIFACE -f -j DROP

# Make sure packets are associated with known connections
$IPT -A ILLEGAL -i $WANIFACE -m state --state INVALID -j DROP

# Make sure NEW tcp connections are SYN packets
$IPT -A ILLEGAL -i $WANIFACE -p tcp ! --syn -m state --state NEW -j DROP

# Refuse bogus IP ranges
$IPT -A ILLEGAL -i $WANIFACE -s 255.255.255.255/32 -j DROP    # Broadcast
$IPT -A ILLEGAL -i $WANIFACE -s 127.0.0.0/8 -j DROP           # Loopback
$IPT -A ILLEGAL -i $WANIFACE -s 169.254.0.0/16 -j DROP        # Link local
networks
$IPT -A ILLEGAL -i $WANIFACE -s 192.0.2.0/24 -j DROP          # Test-net
$IPT -A ILLEGAL -i $WANIFACE -s 248.0.0.0/5 -j DROP           # Unallocated
```

```
$IPT -A ILLEGAL -i $WANIFACE -s 10.0.0.0/8 -j DROP           # Class A private
(RFC 1918)
$IPT -A ILLEGAL -i $WANIFACE -s 172.16.0.0/16 -j DROP        # Class B private
(RFC 1918)
$IPT -A ILLEGAL -i $WANIFACE -s 192.168.0.0/16 -j DROP       # Class C private
(RFC 1918)
$IPT -A ILLEGAL -i $WANIFACE -s 224.0.0.0/4 -j DROP          # Class D
multicast
$IPT -A ILLEGAL -i $WANIFACE -s 240.0.0.0/5 -j DROP          # Class E reserved

$IPT -A INPUT -i $WANIFACE -p tcp -j ILLEGAL
#$IPT -A INPUT -j LOG
}

if [ $BASIC_PROTECTION = "ON" ]; then
   BASIC_PROTECTION
fi

#-----------------------------------------#
#             IANA RESERVED               #
#-----------------------------------------#

function IANA_RESERVED {

# Drop IANA reserved address space (RFC1466, RFC1918, RFC3330)
# This list may need to be updated from time to time.
# http://www.iana.org/assignments/ipv4-address-space
# Last Updated 2003-04-05

RESERVED="0 1 2 5 7 23 27 31 36 37 39 41 42 58 59 70 71 72 73 74 75 76 77 78 79
83 84 85 86 87 88 89 90 91 92 93 94 95 96 97
98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117
118 119 120 121 122 123 124 125 126 127 173 174
 175 176 177 178 179 180 181 182 183 184 185 186 187 189 190 197 223 240 241 242
243 244 245 246 247 248 249 250 251 252 253
254 255"

for NET_LOOP in $RESERVED; do
$IPT -A ILLEGAL -i $WANIFACE -s $NET_LOOP.0.0.0/8 -j DROP
done
}

if [ "$IANA_RESERVED" = "ON" ]; then
   IANA_RESERVED
fi

#-----------------------------------------#
#              Banned List                #
#-----------------------------------------#

function BANNED_IPS {

if [ -e $BANNED_IPS ]; then
  $IPT -N BANNED
  $IPT -F BANNED
    while read BAN; do
      $IPT -A BANNED -s $BAN -j DROP
    done < $BANNED_IPS
  $IPT -A INPUT -j BANNED
else
  echo; echo
  echo "The BANNED_IPS option is enabled but the actual file,"
  echo "\"$BANNED_IPS\" cannot be found."
  echo "Make sure the BANNED_IPS variable includes the full"
  echo "path and filename or set it to NONE to disable."
  echo; echo
```

```
fi
}

if [ "$BANNED_IPS" != "NONE" ]; then
   BANNED_IPS
fi

#----------------------------------------#
#              TOS Tweaks                #
#----------------------------------------#

# (0x00) Normal-Service 0
# (0x02) Minimize-Cost 2
# (0x04) Maximize-Reliability 4
# (0x08) Maximize-Throughput 8
# (0x10) Minimize-Delay 16

function TOS_TWEAKS {

$IPT -t mangle -N MANGLE_OUTPUT
$IPT -t mangle -F MANGLE_OUTPUT

$IPT -t mangle -A MANGLE_OUTPUT -p tcp --dport 20 -j TOS --set-tos 8
$IPT -t mangle -A MANGLE_OUTPUT -p tcp --dport 21 -j TOS --set-tos 16
$IPT -t mangle -A MANGLE_OUTPUT -p tcp --dport 22 -j TOS --set-tos 16
$IPT -t mangle -A MANGLE_OUTPUT -p tcp --dport 25 -j TOS --set-tos 16
$IPT -t mangle -A MANGLE_OUTPUT -p tcp --dport 53 -j TOS --set-tos 16
$IPT -t mangle -A MANGLE_OUTPUT -p udp --dport 53 -j TOS --set-tos 16
$IPT -t mangle -A MANGLE_OUTPUT -p tcp --dport 80 -j TOS --set-tos 8

$IPT -t mangle -N MANGLE_PREROUTING
$IPT -t mangle -F MANGLE_PREROUTING

$IPT -t mangle -A MANGLE_PREROUTING -p tcp --dport 20 -j TOS --set-tos 8
$IPT -t mangle -A MANGLE_PREROUTING -p tcp --dport 21 -j TOS --set-tos 16
$IPT -t mangle -A MANGLE_PREROUTING -p tcp --dport 22 -j TOS --set-tos 16
$IPT -t mangle -A MANGLE_PREROUTING -p tcp --dport 25 -j TOS --set-tos 16
$IPT -t mangle -A MANGLE_PREROUTING -p tcp --dport 53 -j TOS --set-tos 16
$IPT -t mangle -A MANGLE_PREROUTING -p udp --dport 53 -j TOS --set-tos 16
$IPT -t mangle -A MANGLE_PREROUTING -p tcp --dport 80 -j TOS --set-tos 8

$IPT -t mangle -A PREROUTING -i $LANIFACE -j MANGLE_PREROUTING
$IPT -t mangle -A OUTPUT -o $WANIFACE -j MANGLE_OUTPUT
#$IPT -A MANGLE -j LOG
}

if [ $TOS_TWEAKS = "ON" ]; then
   TOS_TWEAKS
fi

################################################################################
###
#                          ALLOWED NETWORK TRAFFIC
#
################################################################################
###

#----------------------------#
#      1 to MANY NAT         #
#----------------------------#

function SNAT {
   echo 0 > /proc/sys/net/ipv4/ip_dynaddr
   echo 1 > /proc/sys/net/ipv4/ip_forward
   $IPT -t nat -A POSTROUTING -s $LAN -o $WANIFACE -j SNAT --to-source $WANIP
}
```

```
function MASQ {
   echo 1 > /proc/sys/net/ipv4/ip_dynaddr
   echo 1 > /proc/sys/net/ipv4/ip_forward
   $IPT -t nat -A POSTROUTING -o $WANIFACE -j MASQUERADE
}

if [ $SNAT = "ON" ] && [ $MASQ = "OFF" ]; then
   SNAT
fi

if [ $SNAT = "OFF" ] && [ $MASQ = "ON" ]; then
   MASQ
fi

if [ $SNAT = "ON" ] && [ $MASQ = "ON" ]; then
   echo;echo
   echo "You cannot enable both the SNAT and MASQ options in the firewall."
   echo "If you have a static IP, then enable the SNAT option only.  If you"
   echo "have a non-static IP, enable the MASQ option only.  Right now, you"
   echo "do not have any form of NAT running.  Check the CONFIGURATION"
   echo "section in the firewall and try again."
   echo;echo
fi

#----------------------------#
#      LOCAL TRAFFIC         #
#----------------------------#

# Allow all existing connections
$IPT -I INPUT 1 -p ALL -m state --state ESTABLISHED,RELATED -j ACCEPT
$IPT -I FORWARD 1 -p ALL -m state --state ESTABLISHED,RELATED -j ACCEPT
$IPT -I OUTPUT 1 -p ALL -m state --state ESTABLISHED,RELATED -j ACCEPT

# Allow all LAN
$IPT -I INPUT 2 -p ALL -i $LANIFACE -s $LAN -j ACCEPT
$IPT -I FORWARD 2 -p ALL -i $LANIFACE -s $LAN -j ACCEPT
$IPT -I OUTPUT 2 -p ALL -o $WANIFACE -j ACCEPT
$IPT -I OUTPUT 3 -p ALL -o $LANIFACE -j ACCEPT
#$IPT -A INPUT -j LOG

# Allow localhost
$IPT -A INPUT -i lo -j ACCEPT
$IPT -A FORWARD -i lo -s $LAN -j ACCEPT
$IPT -A OUTPUT -p ALL -o lo -j ACCEPT

#----------------------------#
#       ICMP TRAFFIC         #
#----------------------------#

#  0 = Echo Reply, what gets sent back after a type 8 is received here
#  3 = Destination Unreachable (inbound) or Fragmentation Needed (out)
#  4 = Source Quench tells sending IP to slow down its rate to destination
#  8 = Echo Request used for pinging hosts, but see the caution above
# 11 = Time Exceeded used for traceroute (TTL) or sometimes frag packets
# 12 = Parameter Problem is some error or weirdness detected in header

function ICMP_WAN {

# Allow all ICMP traffic
$IPT -A INPUT -i $WANIFACE -p ICMP -j ACCEPT
}

function ICMP_LIMIT {

$IPT -N ICMP
$IPT -F ICMP
```

```
# Allow limited ICMP traffic
$IPT -A ICMP -i $WANIFACE -p ICMP --icmp-type 3 -j ACCEPT   # Destination
Unreachable
$IPT -A ICMP -i $WANIFACE -p ICMP --icmp-type 4 -j ACCEPT   # Source Quench
$IPT -A ICMP -i $WANIFACE -p ICMP --icmp-type 11 -j ACCEPT  # Time Exceeded
$IPT -A ICMP -i $WANIFACE -p ICMP --icmp-type 12 -j ACCEPT  # Parameter Problem

$IPT -A INPUT -i $WANIFACE -p icmp -j ICMP
}

if [ "$ICMP_TRAFFIC" = "ON" ]; then
   ICMP_WAN
fi

if [ "$ICMP_TRAFFIC" = "LIMIT" ]; then
   ICMP_LIMIT
fi

#-----------------------------#
#      UNCOMMON PORTS         #
#-----------------------------#

if [ "$TCP_PORTS" != "OFF" ]; then
$IPT -A INPUT -p tcp -i $WANIFACE --dport $TCP_PORTS -j ACCEPT
fi

if [ "$UDP_PORTS" != "OFF" ]; then
$IPT -A INPUT -p udp -i $WANIFACE --dport $UDP_PORTS -j ACCEPT
fi

#------ UT2k ---------
$IPT -A INPUT -p udp -i $WANIFACE --dport 7777:7800 -j ACCEPT
#$IPT -A INPUT -p udp -i $WANIFACE --dport 7778 -j ACCEPT
#$IPT -A INPUT -p udp -i $WANIFACE --dport 7787 -j ACCEPT

$IPT -A INPUT -p tcp -i $WANIFACE --dport 28902 -j ACCEPT
$IPT -A INPUT -p tcp -i $WANIFACE --dport 10001 -j ACCEPT
#-----------------------------#
#           FTP               #
#-----------------------------#

function FTP_WAN {
   $IPT -A INPUT -p tcp -i $WANIFACE --dport 20 -j ACCEPT
   $IPT -A INPUT -p tcp -i $WANIFACE --dport 21 -j ACCEPT
}

function FTP_PORT_FORWARDING {
   $IPT -A PREROUTING -t nat -i $WANIFACE -p tcp --dport 20 -j DNAT --to $FTP:20
   $IPT -A PREROUTING -t nat -i $WANIFACE -p tcp --dport 21 -j DNAT --to $FTP:21
   $IPT -A FORWARD -i $WANIFACE -p tcp --dport 20 -d $FTP -j ACCEPT
   $IPT -A FORWARD -i $WANIFACE -p tcp --dport 21 -d $FTP -j ACCEPT
}

if [ $FTP = "ON" ]; then
   FTP_WAN
else
   if [ "$FTP" != "OFF"  ]; then
      FTP_PORT_FORWARDING
   fi
fi

#-----------------------------#
#           WEBMIN            #
#-----------------------------#

# $IPT -A INPUT -p tcp -i $WANIFACE --dport 10000 -j ACCEPT
```

```
function WEBMIN_WAN {
 $IPT -A INPUT -p tcp -s 80.109.144.142 --dport 10000 -j ACCEPT
 $IPT -A INPUT -p tcp -s 80.109.144.140 --dport 10000 -j ACCEPT
}

function WEBMIN_PORT_FORWARDING {
 $IPT -A PREROUTING -t nat -i $WANIFACE -p tcp --dport 10000 -j DNAT --to
$WEBMIN:10000
 $IPT -A FORWARD -i $WANIFACE -p tcp --dport 10000 -j ACCEPT
}
if [ $WEBMIN = "ON" ]; then
 WEBMIN_WAN
  else
   if [ "$WEBMIN" != "OFF"  ]; then
    WEBMIN_PORT_FORWARDING
  fi
 fi

#-----------------------------#
#           SNMP              #
#-----------------------------#

# $IPT -A INPUT -p tcp -i $WANIFACE --dport 10000 -j ACCEPT

function SNMP_WAN {

 echo "ACCEPT EDONKEY";

 $IPT -A INPUT -p udp -i $WANIFACE --dport 161 -j ACCEPT
 $IPT -A INPUT -p udp -i $WANIFACE --dport 162 -j ACCEPT

}

function SNMP_PORT_FORWARDING {
 $IPT -A PREROUTING -t nat -i $WANIFACE -p udp --dport 161 -j DNAT --to
$SNMP:161
 $IPT -A FORWARD -i $WANIFACE -p udp --dport 161 -j ACCEPT
 $IPT -A PREROUTING -t nat -i $WANIFACE -p udp --dport 162 -j DNAT --to
$SNMP:162
 $IPT -A FORWARD -i $WANIFACE -p udp --dport 162 -j ACCEPT
}

if [ $SNMP = "ON" ]; then
 SNMP_WAN
  else
   if [ "$SNMP" != "OFF"  ]; then
    SNMP_PORT_FORWARDING
  fi
 fi

#-----------------------------#
#           SSH               #
#-----------------------------#

function SSH_WAN {
#$IPT -A INPUT -p tcp -s 80.109.144.142 --dport 22 -j ACCEPT
#$IPT -A INPUT -p tcp -s 80.109.144.140 --dport 22 -j ACCEPT
$IPT -A INPUT -p tcp -i $WANIFACE --dport 22 -j ACCEPT
$IPT -A INPUT -p udp -i $WANIFACE --dport 22 -j ACCEPT

}

function SSH_PORT_FORWARDING {
   $IPT -A PREROUTING -t nat -i $WANIFACE -p tcp --dport 22 -j DNAT --to $SSH:22
   $IPT -A FORWARD -i $WANIFACE -p tcp --dport 22 -j ACCEPT
}
```

```
if [ $SSH = "ON" ]; then
   SSH_WAN
else
   if [ "$SSH" != "OFF"  ]; then
      SSH_PORT_FORWARDING
   fi
fi

#----------------------------#
#          SAMBA             #
#----------------------------#

function SAMBA_WAN {
$IPT -A INPUT -p tcp -s $LAN --dport 135:139 -j ACCEPT
$IPT -A INPUT -p udp -s $LAN --dport 135:139 -j ACCEPT
#$IPT -A INPUT -p tcp -i $WANIFACE --dport 22 -j ACCEPT

#$IPT -A INPUT -p tcp -i $LANIFACE --dport 135 -j ACCEPT
#$IPT -A INPUT -p tcp -i $LANIFACE --dport 135 -j ACCEPT
#$IPT -A INPUT -p tcp -i $LANIFACE --dport 136 -j ACCEPT
#$IPT -A INPUT -p tcp -i $LANIFACE --dport 136 -j ACCEPT

#$IPT -A INPUT -p tcp -i $LANIFACE --dport 137 -j ACCEPT
#$IPT -A INPUT -p tcp -i $LANIFACE --dport 137 -j ACCEPT
#$IPT -A INPUT -p tcp -i $LANIFACE --dport 138 -j ACCEPT
#$IPT -A INPUT -p tcp -i $LANIFACE --dport 138 -j ACCEPT
#$IPT -A INPUT -p tcp -i $LANIFACE --dport 139 -j ACCEPT
#$IPT -A INPUT -p tcp -i $LANIFACE --dport 139 -j ACCEPT
#$IPT -A INPUT -p udp -i $LANIFACE --dport 137 -j ACCEPT
#$IPT -A INPUT -p udp -i $LANIFACE --dport 137 -j ACCEPT
#$IPT -A INPUT -p udp -i $LANIFACE --dport 138 -j ACCEPT
#$IPT -A INPUT -p udp -i $LANIFACE --dport 138 -j ACCEPT
#$IPT -A INPUT -p udp -i $LANIFACE --dport 139 -j ACCEPT
#$IPT -A INPUT -p udp -i $LANIFACE --dport 139 -j ACCEPT

#@{CHANGE

#@CHANGE}

}

function SAMBA_PORT_FORWARDING {
   $IPT -A PREROUTING -t nat -i $WANIFACE -p tcp --dport 135 -j DNAT --to
$SAMBA:135
   $IPT -A FORWARD -i $WANIFACE -p tcp --dport 135 -j ACCEPT
   $IPT -A PREROUTING -t nat -i $WANIFACE -p udp --dport 135 -j DNAT --to
$SAMBA:135
   $IPT -A FORWARD -i $WANIFACE -p udp --dport 135 -j ACCEPT

   $IPT -A PREROUTING -t nat -i $WANIFACE -p tcp --dport 136 -j DNAT --to
$SAMBA:136
   $IPT -A FORWARD -i $WANIFACE -p tcp --dport 136 -j ACCEPT
   $IPT -A PREROUTING -t nat -i $WANIFACE -p udp --dport 136 -j DNAT --to
$SAMBA:136
   $IPT -A FORWARD -i $WANIFACE -p udp --dport 135 -j ACCEPT
}

if [ $SAMBA = "ON" ]; then
   SAMBA_WAN
else
   if [ "$SAMBA" != "OFF"  ]; then
      SAMBA_PORT_FORWARDING
   fi
fi
```

```
#----------------------------#
#          SMTP              #
#----------------------------#

function SMTP_WAN {
   $IPT -A INPUT -p tcp -i $WANIFACE --dport 25 -j ACCEPT
   $IPT -A INPUT -p udp -i $WANIFACE --dport 25 -j ACCEPT
}

function SMTP_PORT_FORWARDING {
   $IPT -A PREROUTING -t nat -i $WANIFACE -p tcp --dport 25 -j DNAT --to
$SMTP:25
   $IPT -A FORWARD -i $WANIFACE -p tcp --dport 25 -j ACCEPT
}

if [ $SMTP = "ON" ]; then
   SMTP_WAN
else
   if [ "$SMTP" != "OFF"  ]; then
      SMTP_PORT_FORWARDING
   fi
fi

#----------------------------#
#           DNS              #
#----------------------------#

function DNS_WAN {
   $IPT -A INPUT -i $WANIFACE -p udp --dport 53 -s 0/0 -j ACCEPT
}

function DNS_PORT_FORWARDING {
   $IPT -A PREROUTING -t nat -i $WANIFACE -p udp --dport 53 -j DNAT --to $DNS:53
   $IPT -A FORWARD -i $WANIFACE -p udp --dport 53 -j ACCEPT
}

if [ $DNS = "ON" ]; then
   DNS_WAN
else
   if [ "$DNS" != "OFF"  ]; then
      DNS_PORT_FORWARDING
   fi
fi


#----------------------------#
#          TFTP              #
#----------------------------#

function TFTP_WAN {
   $IPT -A INPUT -p tcp -i $WANIFACE --dport 69 -j ACCEPT
}

function TFTP_PORT_FORWARDING {
   $IPT -A PREROUTING -t nat -i $WANIFACE -p tcp --dport 69 -j DNAT --to
$TFTP:69
   $IPT -A FORWARD -i $WANIFACE -p tcp --dport 69 -j ACCEPT
}

if [ $TFTP = "ON" ]; then
   TFTP_WAN
else
   if [ "$TFTP" != "OFF"  ]; then
      TFTP_PORT_FORWARDING
   fi
fi
```

```
#---------------------------#
#           HTTP            #
#---------------------------#

function HTTP {
   $IPT -A INPUT -p tcp -i $WANIFACE --dport 80 -j ACCEPT
}

function HTTP_PORT_FORWARDING {
   $IPT -A PREROUTING -t nat -i $WANIFACE -p tcp --dport 80 -j DNAT --to
$HTTP:80
   $IPT -A FORWARD -i $WANIFACE -p tcp --dport 80 -j ACCEPT
}

if [ $HTTP = "ON" ]; then
   HTTP
else
   if [ "$HTTP" != "OFF"  ]; then
      HTTP_PORT_FORWARDING
   fi
fi

#---------------------------#
#           AUTH            #
#---------------------------#

function AUTH_WAN {
   $IPT -A INPUT -p tcp -i $WANIFACE --dport 113 -j ACCEPT
}

function AUTH_PORT_FORWARDING {
   $IPT -A PREROUTING -t nat -i $WANIFACE -p tcp --dport 113 -j DNAT --to
$AUTH:113
   $IPT -A FORWARD -i $WANIFACE -p tcp --dport 113 -j ACCEPT
}

if [ $AUTH = "ON" ]; then
   AUTH_WAN
else
   if [ "$AUTH" != "OFF"  ]; then
      AUTH_PORT_FORWARDING
   fi
fi

#---------------------------#
#           POP3            #
#---------------------------#

function POP3_WAN {
   $IPT -A INPUT -p tcp -i $WANIFACE --dport 110 -j ACCEPT
}

function POP3_PORT_FORWARDING {
   $IPT -A PREROUTING -t nat -i $WANIFACE -p tcp --dport 110 -j DNAT --to
$POP3:110
   $IPT -A FORWARD -i $WANIFACE -p tcp --dport 110 -j ACCEPT
}

if [ $POP3 = "ON" ]; then
   POP3_WAN
else
   if [ "$POP3" != "OFF"  ]; then
      POP3_PORT_FORWARDING
   fi
fi
```

```
#----------------------------#
#           NNTP             #
#----------------------------#

function NNTP_WAN {
   $IPT -A INPUT -p tcp -i $WANIFACE --dport 119 -j ACCEPT
}

function NNTP_PORT_FORWARDING {
   $IPT -A PREROUTING -t nat -i $WANIFACE -p tcp --dport 119 -j DNAT --to
$NNTP:119
   $IPT -A FORWARD -i $WANIFACE -p tcp --dport 119 -j ACCEPT
}

if [ $NNTP = "ON" ]; then
   NNTP_WAN
else
   if [ "$NNTP" != "OFF"  ]; then
      NNTP_PORT_FORWARDING
   fi
fi

#----------------------------#
#           NTP              #
#----------------------------#

unction NTP_WAN {
   $IPT -A INPUT -p udp -i $WANIFACE --dport 123 -j ACCEPT
}

function NTP_PORT_FORWARDING {
   $IPT -A PREROUTING -t nat -i $WANIFACE -p udp --dport 123 -j DNAT --to
$NTP:123
   $IPT -A FORWARD -i $WANIFACE -p udp --dport 123 -j ACCEPT
}

if [ $NTP = "ON" ]; then
   NTP_WAN
else
   if [ "$NTP" != "OFF"  ]; then
      NTP_PORT_FORWARDING
   fi
fi

#----------------------------#
#           IMAP4            #
#----------------------------#

function IMAP4_WAN {
   $IPT -A INPUT -p tcp -i $WANIFACE --dport 143 -j ACCEPT
}

function IMAP4_PORT_FORWARDING {
   $IPT -A PREROUTING -t nat -i $WANIFACE -p tcp --dport 143 -j DNAT --to
$IMAP4:143
   $IPT -A FORWARD -i $WANIFACE -p tcp --dport 143 -j ACCEPT
}

if [ $IMAP4 = "ON" ]; then
   IMAP4_WAN
else
   if [ "$IMAP4" != "OFF"  ]; then
      IMAP4_PORT_FORWARDING
   fi
fi
```

```
#---------------------------#
#           LDAP            #
#---------------------------#

function LDAP_WAN {
   $IPT -A INPUT -p tcp -i $WANIFACE --dport 389 -j ACCEPT
}

function LDAP_PORT_FORWARDING {
   $IPT -A PREROUTING -t nat -i $WANIFACE -p tcp --dport 389 -j DNAT --to
$LDAP:389
   $IPT -A FORWARD -i $WANIFACE -p tcp --dport 389 -j ACCEPT
}

if [ $LDAP = "ON" ]; then
   LDAP_WAN
else
   if [ "$LDAP" != "OFF"  ]; then
      LDAP_PORT_FORWARDING
   fi
fi

#---------------------------#
#           SSL             #
#---------------------------#

function SSL_WAN {
   $IPT -A INPUT -p tcp -i $WANIFACE --dport 443 -j ACCEPT
}

function SSL_PORT_FORWARDING {
   $IPT -A PREROUTING -t nat -i $WANIFACE -p tcp --dport 443 -j DNAT --to
$SSL:443
   $IPT -A FORWARD -i $WANIFACE -p tcp --dport 443 -j ACCEPT
}

if [ $SSL = "ON" ]; then
   SSL_WAN
else
   if [ "$SSL" != "OFF"  ]; then
      SSL_PORT_FORWARDING
   fi
fi

#---------------------------#
#       Squid Proxy         #
#---------------------------#

function SQUID_PROXY {
echo 1 > /proc/sys/net/ipv4/conf/all/send_redirects
$IPT -t nat -A PREROUTING -i $LANIFACE -s $LAN -p tcp --dport 80 -j REDIRECT --
to-port 3128
}

function SQUID_PORT_FORWARDING {
   $IPT -t nat -A PREROUTING -i $LANIFACE -s $LAN -p tcp --dport 80 -j REDIRECT
--to $SQUID:3128
}

if [ $SQUID = "ON" ]; then
   SQUID_PROXY
else
   if [ "$SQUID" != "OFF"  ]; then
      SQUID_PORT_FORWARDING
   fi
fi
```

```
#################################################################################
###
#                          FIREWALL LOADING FUNCTIONS
#
#################################################################################
###

echo "<< SYSTEM IS PROTECTED >>  All tables loaded and policies in place"
}

 {
   if [ -f $IPT ]; then
      CHAINS=`cat /proc/net/ip_tables_names 2>/dev/null`
      for TABLE in $CHAINS; do $IPT -t $TABLE -F; done; $IPT -F
      for TABLE in $CHAINS; do $IPT -t $TABLE -X; done; $IPT -X
      for TABLE in $CHAINS; do $IPT -t $TABLE -Z; done; $IPT -Z
   fi

   CHAINS="INPUT FORWARD OUTPUT"
   for TABLE in $CHAINS; do $IPT -P $TABLE ACCEPT; done
   CHAINS="PREROUTING POSTROUTING OUTPUT"
   for TABLE in $CHAINS; do $IPT -t nat -P $TABLE ACCEPT; done
   CHAINS="PREROUTING OUTPUT"
   for TABLE in $CHAINS; do $IPT -t mangle -P $TABLE ACCEPT; done
   echo 0 > /proc/sys/net/ipv4/ip_forward
   echo 0 > /proc/sys/net/ipv4/ip_dynaddr
   rm -f /var/lock/subsys/iptables
   echo "<< SYSTEM IS OPEN >>  All tables flushed and policies set to ACCEPT"
}

firewall_lock() {
   firewall_down
   CHAINS="INPUT FORWARD OUTPUT"
   for TABLE in $CHAINS; do $IPT -P $TABLE DROP; done
   CHAINS="PREROUTING POSTROUTING OUTPUT"
   for TABLE in $CHAINS; do $IPT -t nat -P $TABLE DROP; done
   CHAINS="PREROUTING OUTPUT"
   for TABLE in $CHAINS; do $IPT -t mangle -P $TABLE DROP; done
   echo "<< SYSTEM IS LOCKED DOWN >>  All tables flushed and policies set to
DROP"
}

firewall_reload() {
   firewall_down;
   firewall_up
}

firewall_list_all() {
   clear
   $IPT -L -v -n --line-numbers|more
}

firewall_list() {
   clear
   echo; echo
   $IPT -L INPUT -v -n --line-numbers|more
   echo; echo
   $IPT -L OUTPUT -v -n --line-numbers|more
   echo; echo
   $IPT -L FORWARD -v -n --line-numbers|more
   echo; echo
   echo " << NETWORK ADDRESS TRANSLATION >>"
   echo; echo
   $IPT -t nat -L -v -n --line-numbers|more
   echo; echo
}
```

```
firewall_mangle() {
   clear
   $IPT -L -t mangle --line-numbers|more
}

firewall_nat() {
   clear
   $IPT -t nat -L -v -n --line-numbers|more
}

firewall_traffic() {
   clear
   cat /proc/net/ip_conntrack|more
}

case "$1" in
'up')
  firewall_up
  ;;
'down')
  firewall_down
  ;;
'lock')
  firewall_lock
  ;;
'reload')
  firewall_reload
  ;;
'list')
  firewall_list
  ;;
'listall')
  firewall_list_all
  ;;
'mangle')
  firewall_mangle
  ;;
'nat')
 firewall_nat
  ;;
'traffic')
 firewall_traffic
  ;;
*)

echo "usage $0 up|down|lock|reload|list|listall|mangle|nat|traffic"

esac
```

[38]

Anhang C – Inhalt der CD-ROM

Die beigefügte CD-Rom enthält die 4 Ordner: Kernel , Programmierte Sicherheitswerkzeuge, Weitere Sicherheitswerkzeuge und Sicherheitspatches.
Im Ordner „Kernelsicherheit“ befindet sich der aktuellste Kernel der 2.6 er Revision mit den vorkompilierten Patches Grsecurity und SE-Linux.
Im Ordner „Programmierte Sicherheitswerkzeuge“ befinden sich die im Rahmen dieser Arbeit programmierten Perl Scripte mit entsprechend dokumentiertem Quellcode.

Der Ordner „Weitere Sicherheitswerkzeuge“ enthält die Programme Nemesis, Nmap und Nessus, Portsentry, TCP-Wrapper, ModSecurity, Snort und die Firewall als Quellcode.
Im letzen Ordner „Sicherheitspatches“ befinden sich Grsecurity mit dem IPTables Patch und Libsafe.

Danksagung

Ich möchte hiermit die Chance nutzen, mich bei meinen Tutoren Herrn Prof. Dr.-Ing. Habil. Joachim Geiler und Herrn Dipl.-Inf. (FH) Matthias Lühr für die Zusammenarbeit zu bedanken. Weiterhin danke ich meiner Familie, die mich beim Schreiben dieser Arbeit unterstützt hat.